Particle

MESH COMMUNICATION AND PARTICLE PRIMITIVES

GETTING STARTED WITH THE WEB IDE

MESH PUBLISH & SUBSCRIBE

WORKING WITH PARTICLE PRIMITIVES

MANAGING DEVICES FROM THE CONSOLE

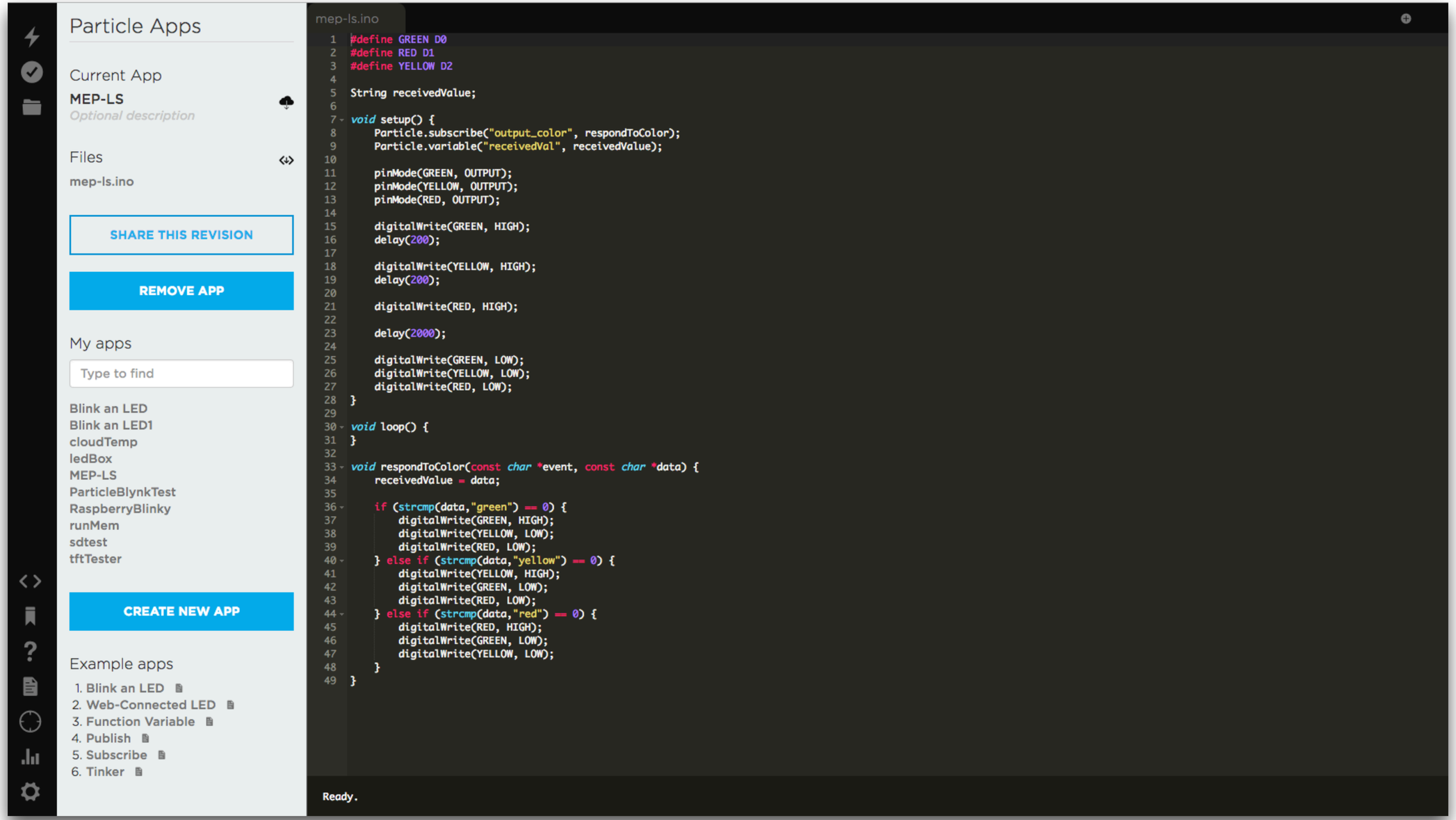
GETTING STARTED WITH THE WEB IDE

MESH PUBLISH & SUBSCRIBE

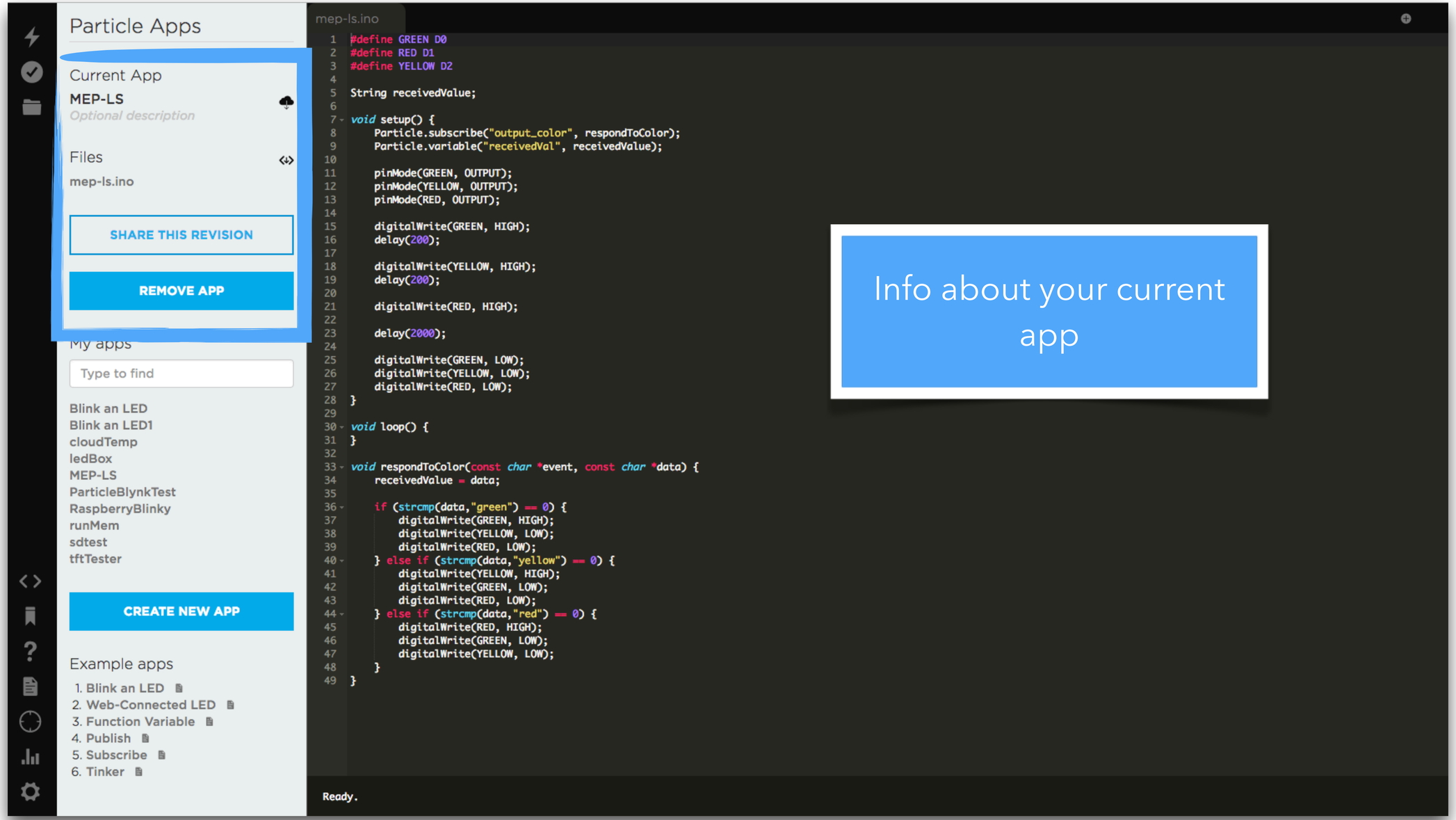
WORKING WITH PARTICLE PRIMITIVES

MANAGING DEVICES FROM THE CONSOLE

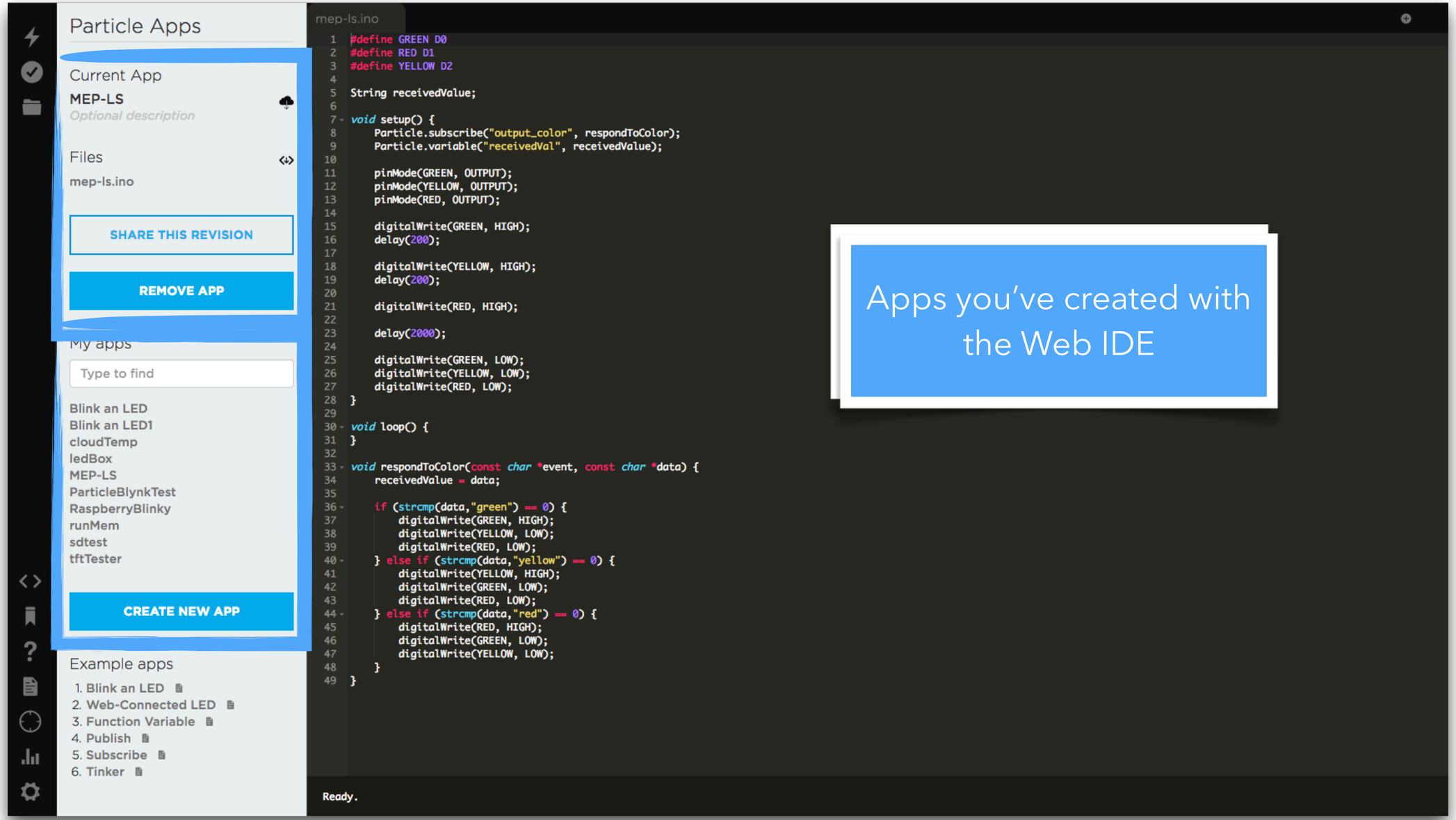
BUILD.PARTICLE.IO (WEB IDE)



BUILD.PARTICLE.IO (WEB IDE)

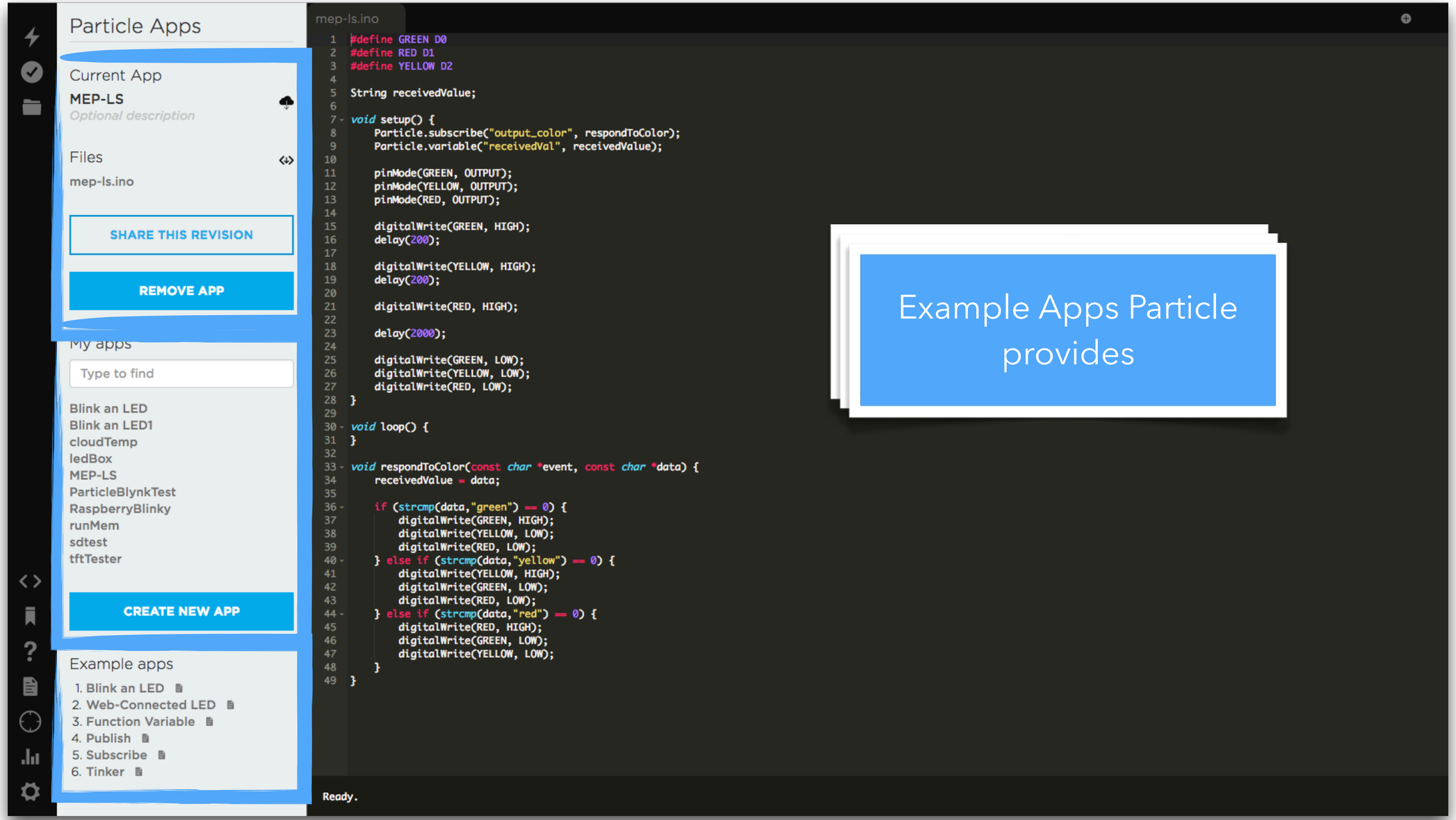


BUILD.PARTICLE.IO (WEB IDE)



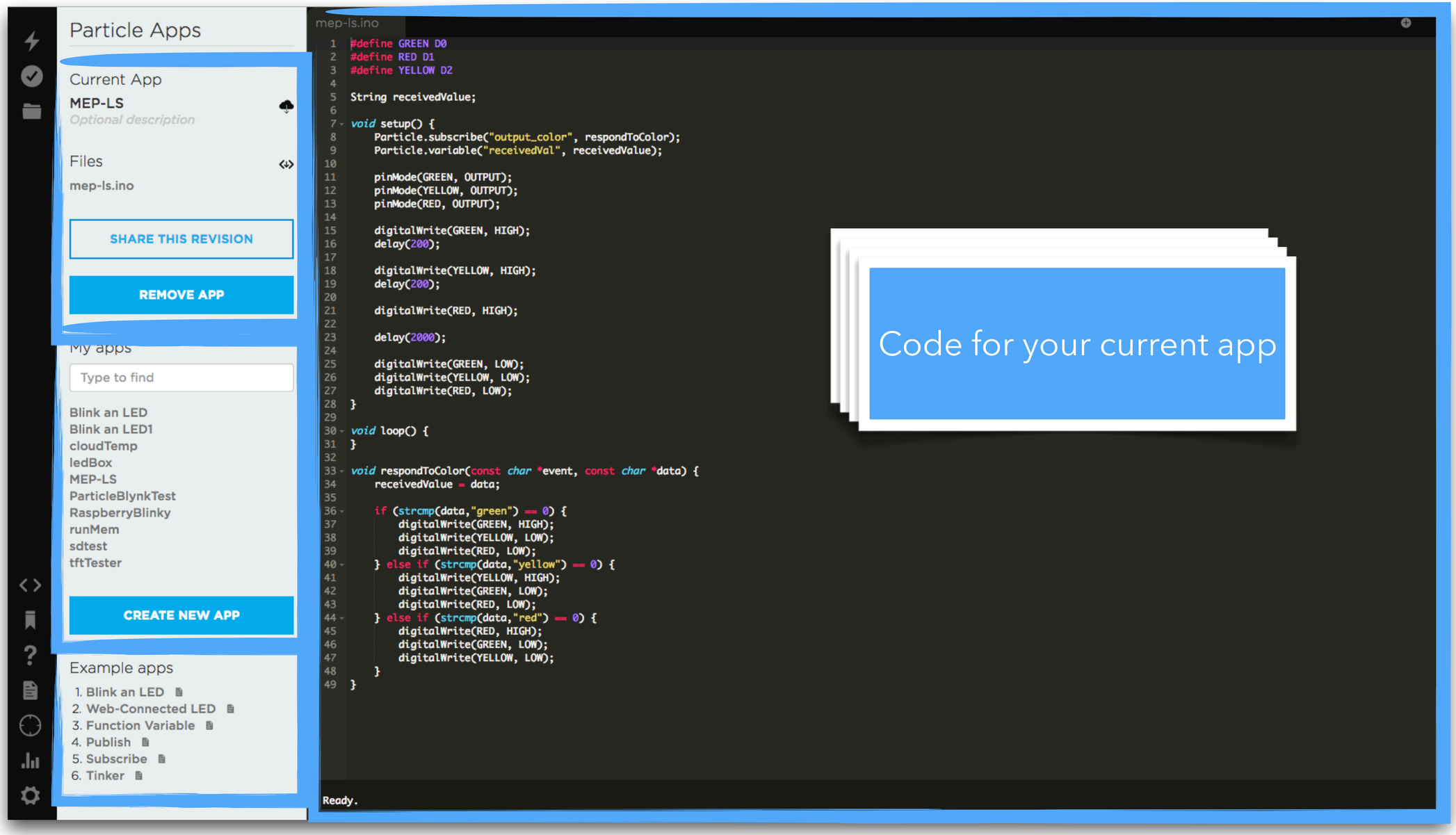
Apps you've created with the Web IDE

BUILD.PARTICLE.IO (WEB IDE)

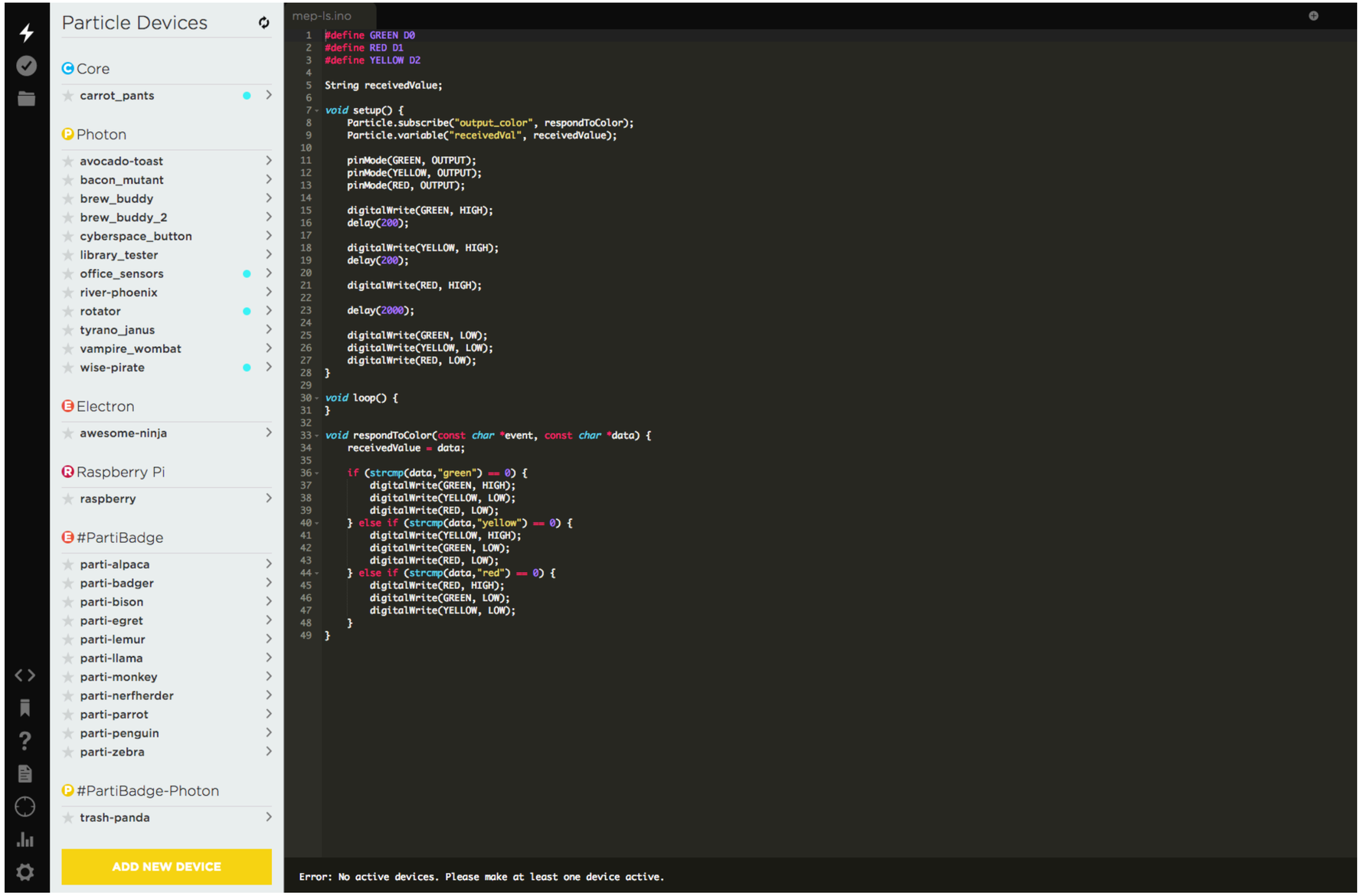


Example Apps Particle provides

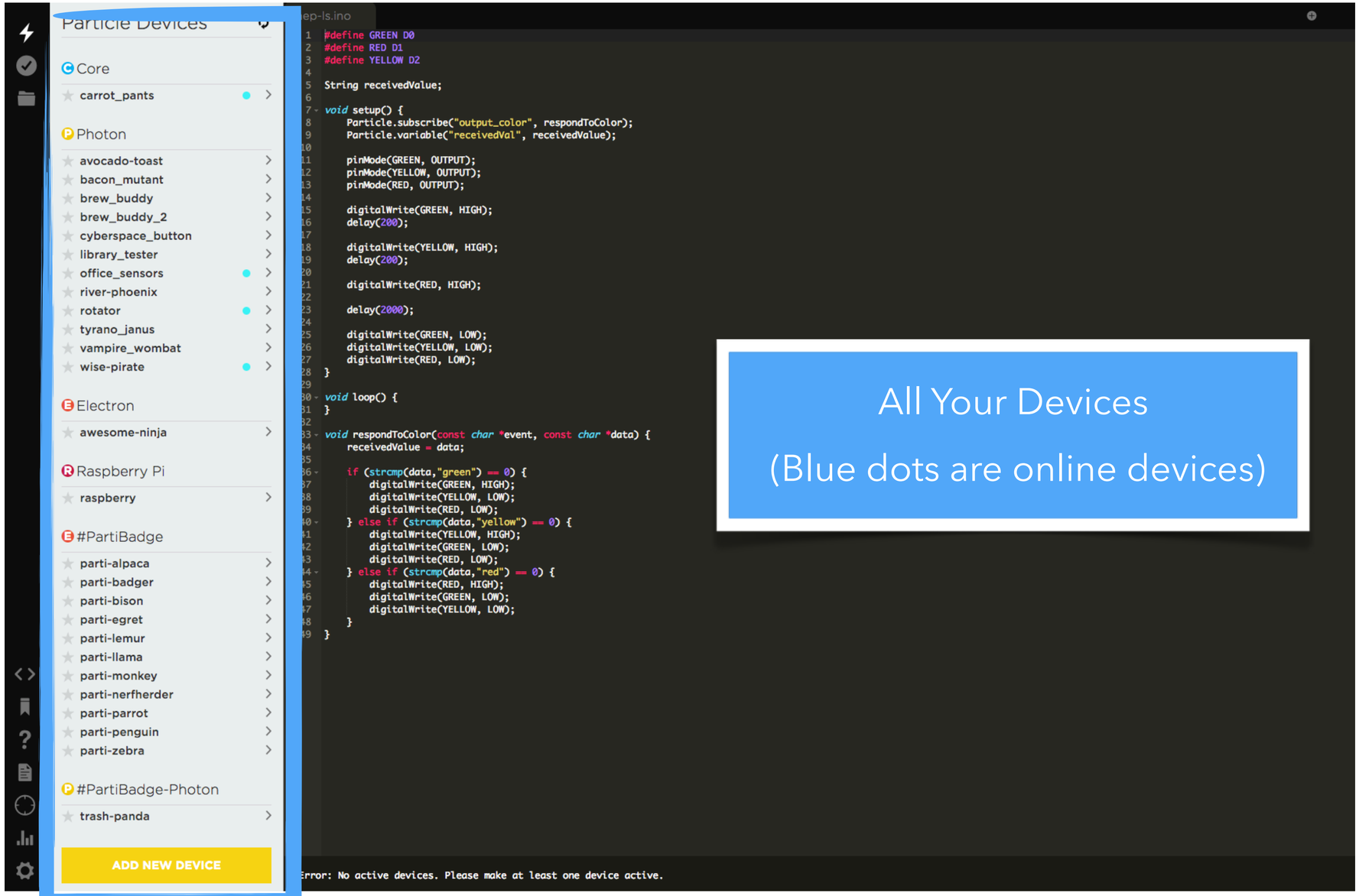
BUILD.PARTICLE.IO (WEB IDE)

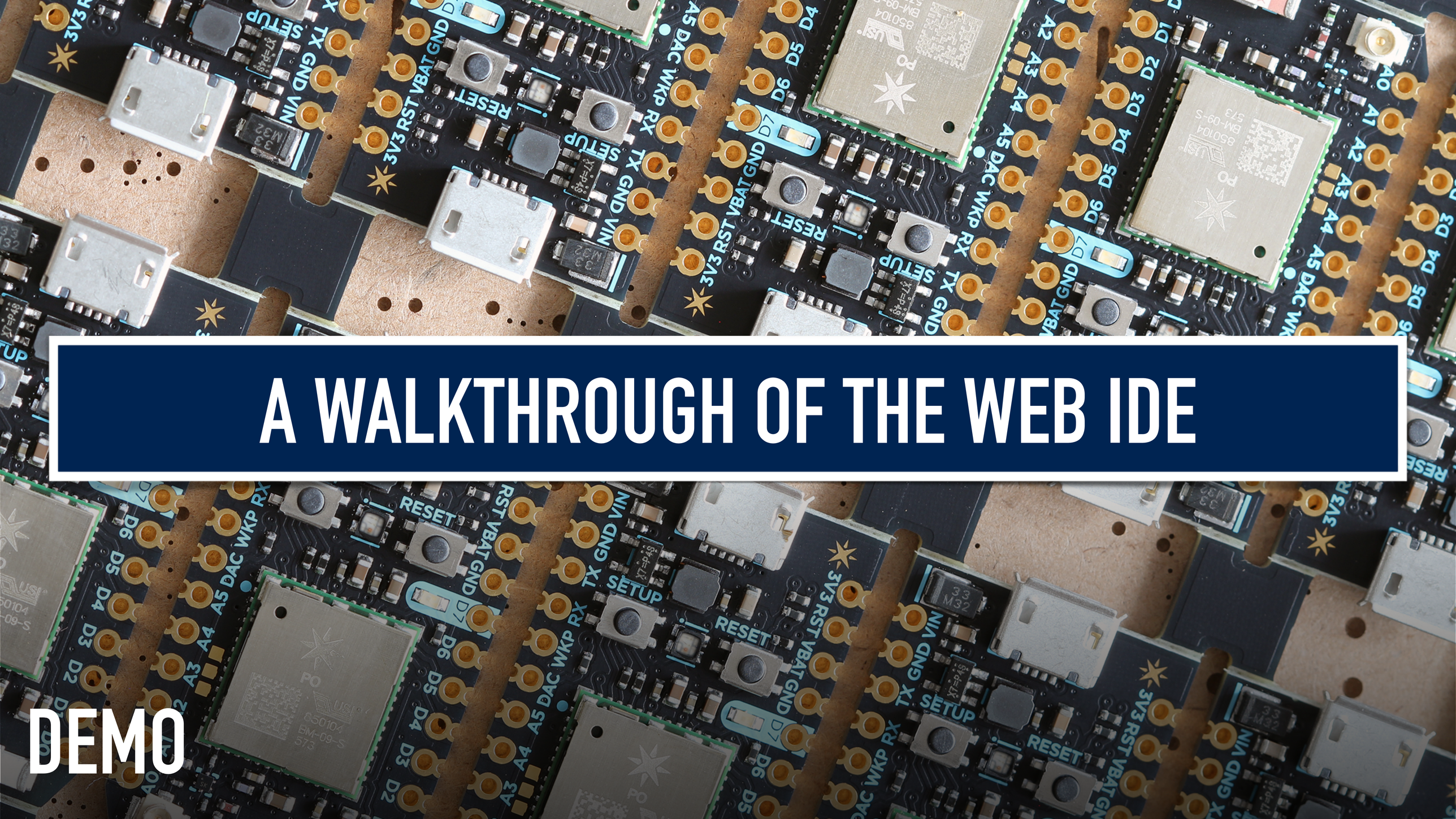


BUILD.PARTICLE.IO (WEB IDE)



BUILD.PARTICLE.IO (WEB IDE)




A WALKTHROUGH OF THE WEB IDE
DEMO

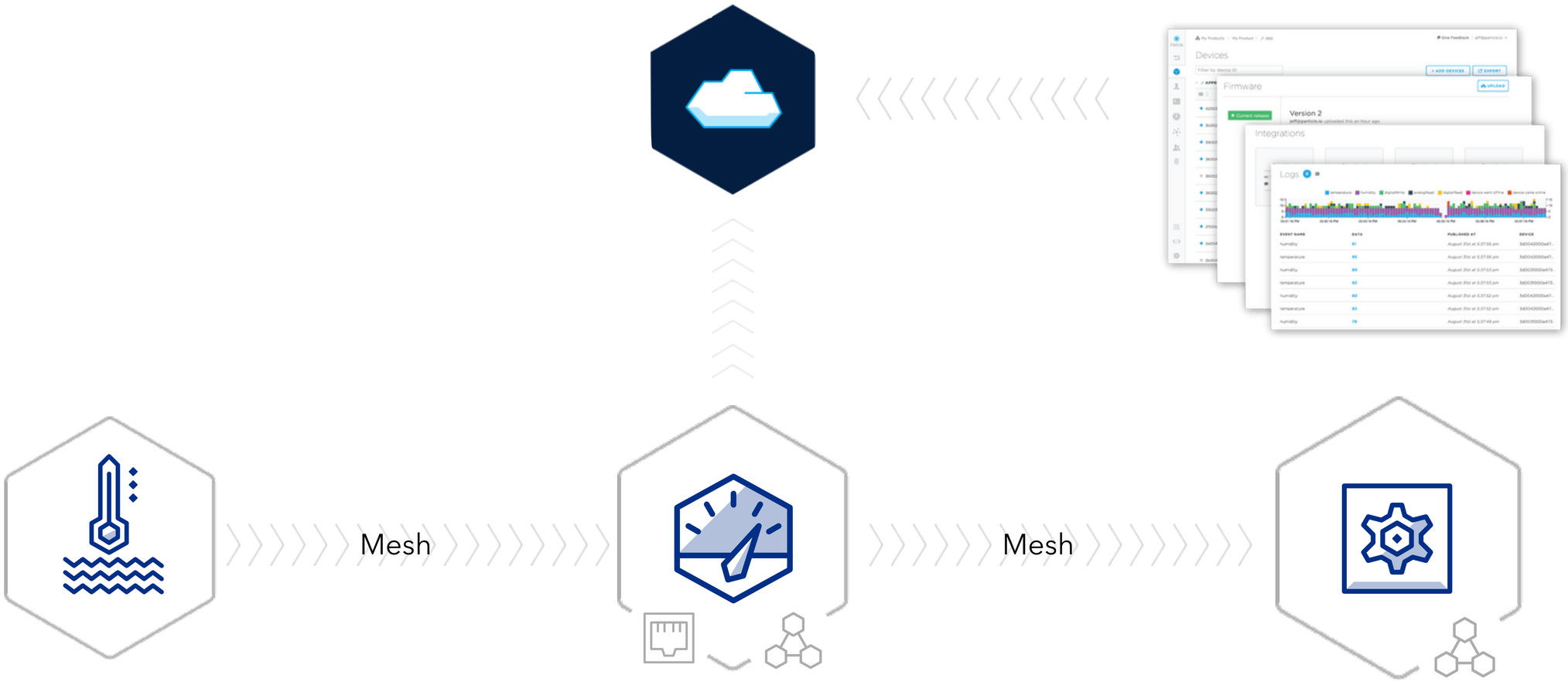
GETTING STARTED WITH THE WEB IDE

MESH PUBLISH & SUBSCRIBE

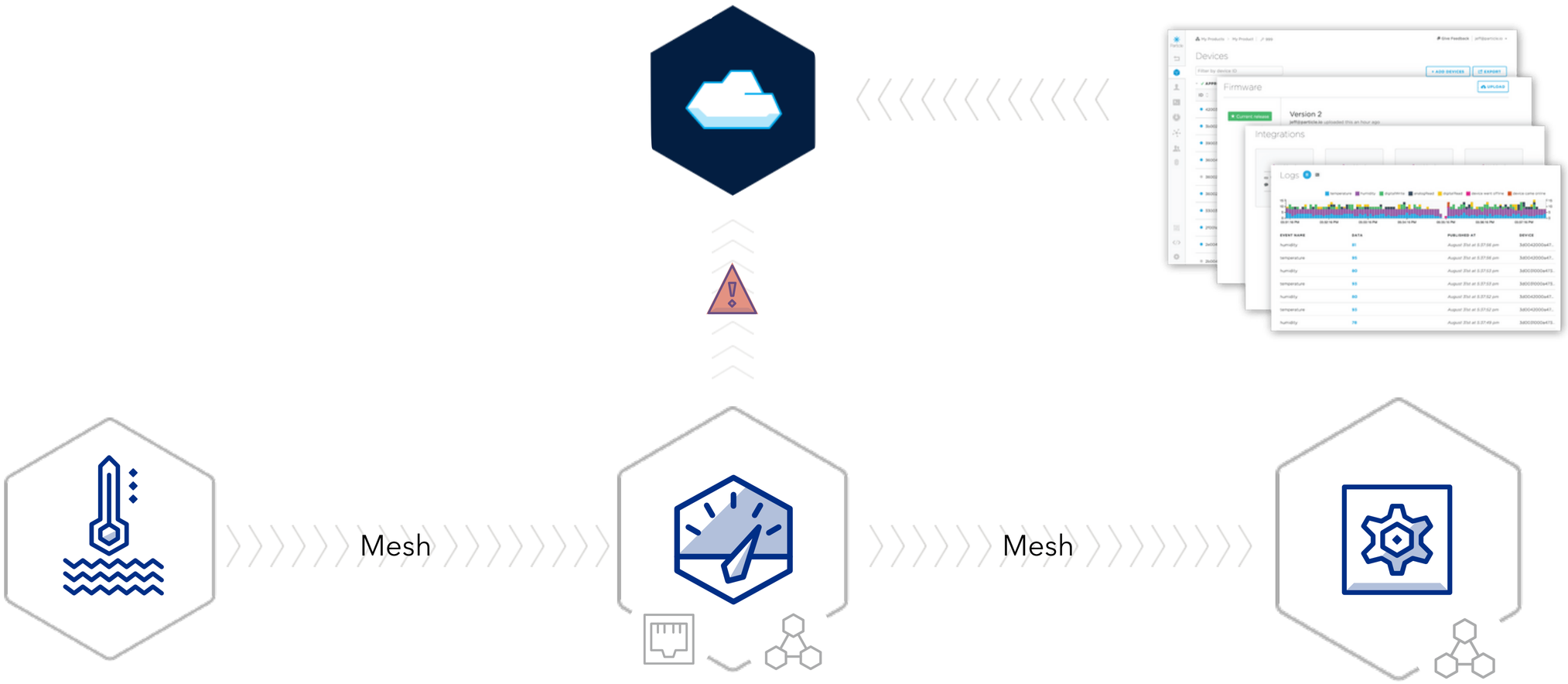
WORKING WITH PARTICLE PRIMITIVES

MANAGING DEVICES FROM THE CONSOLE

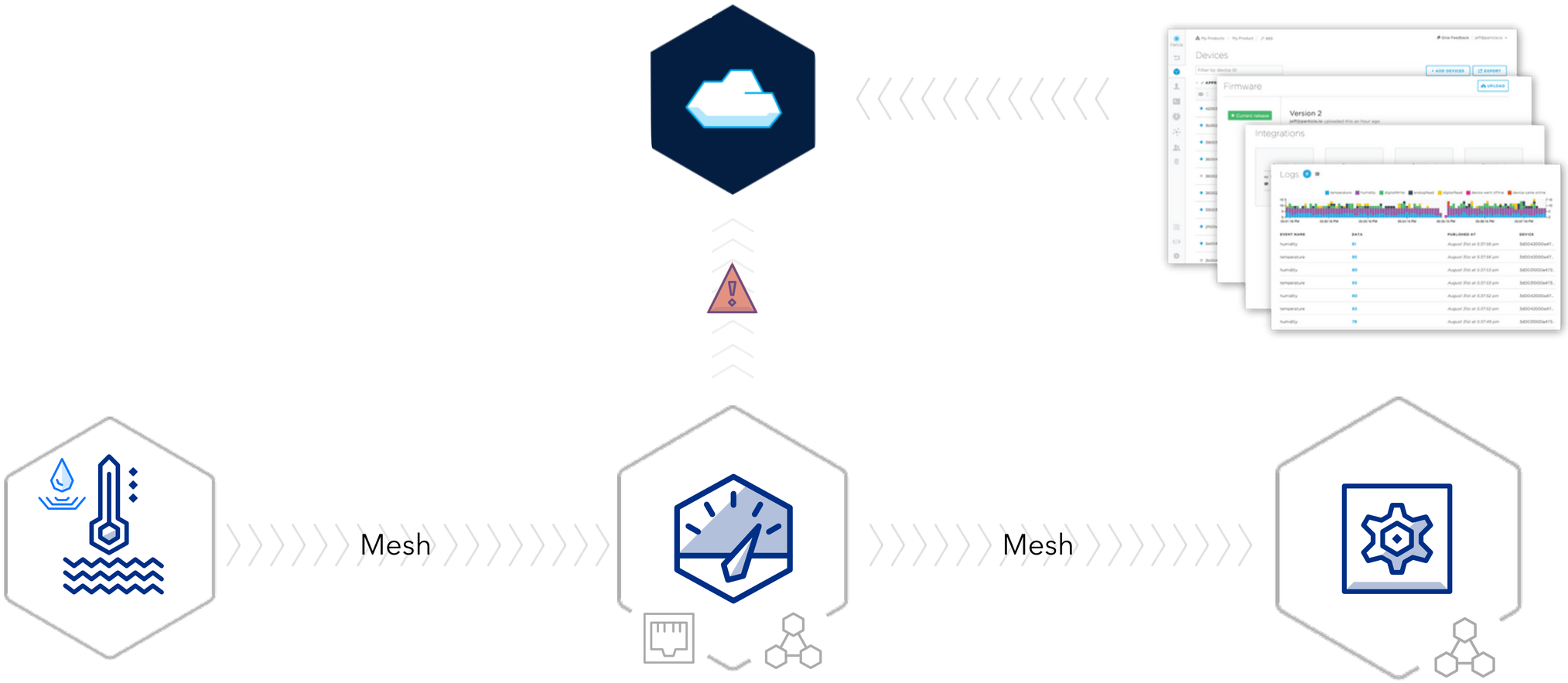
PUB/SUB WITH PARTICLE MESH



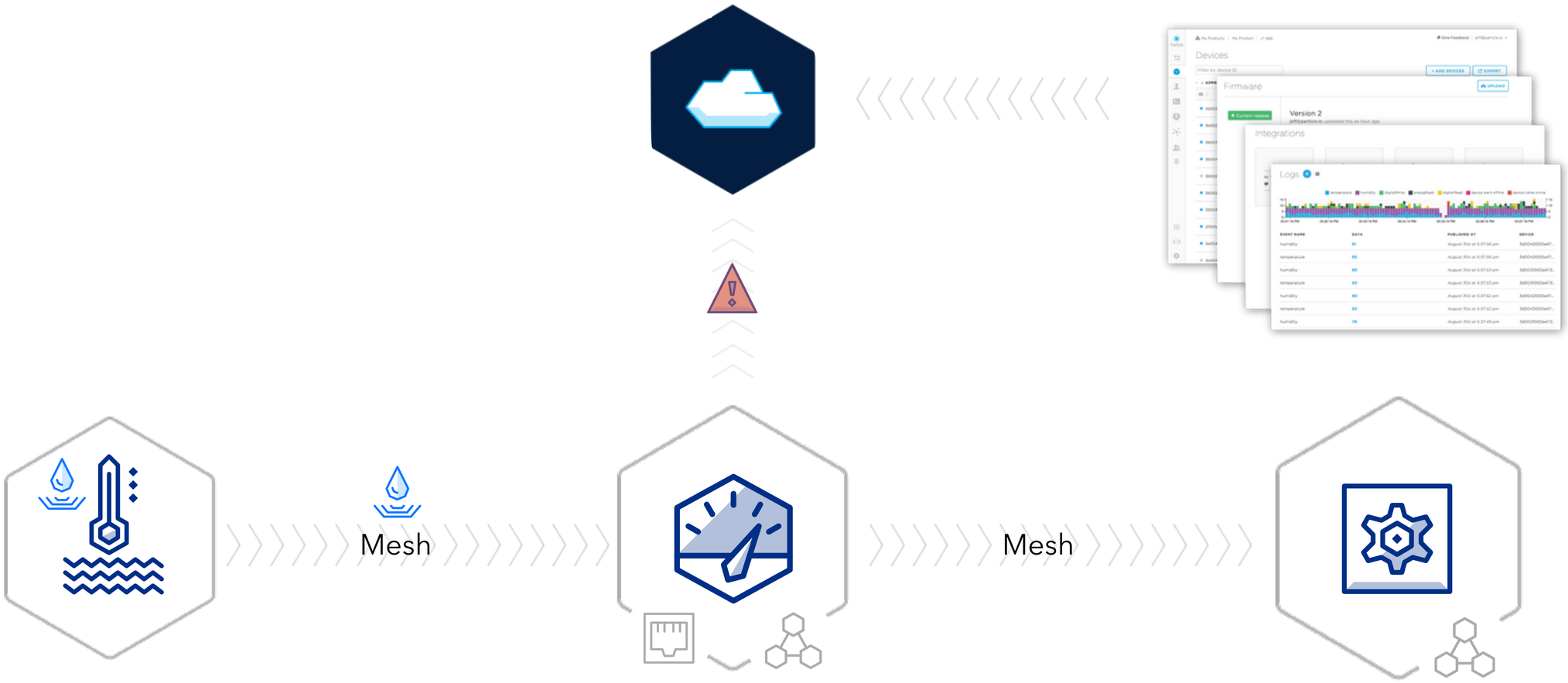
PUB/SUB WITH PARTICLE MESH



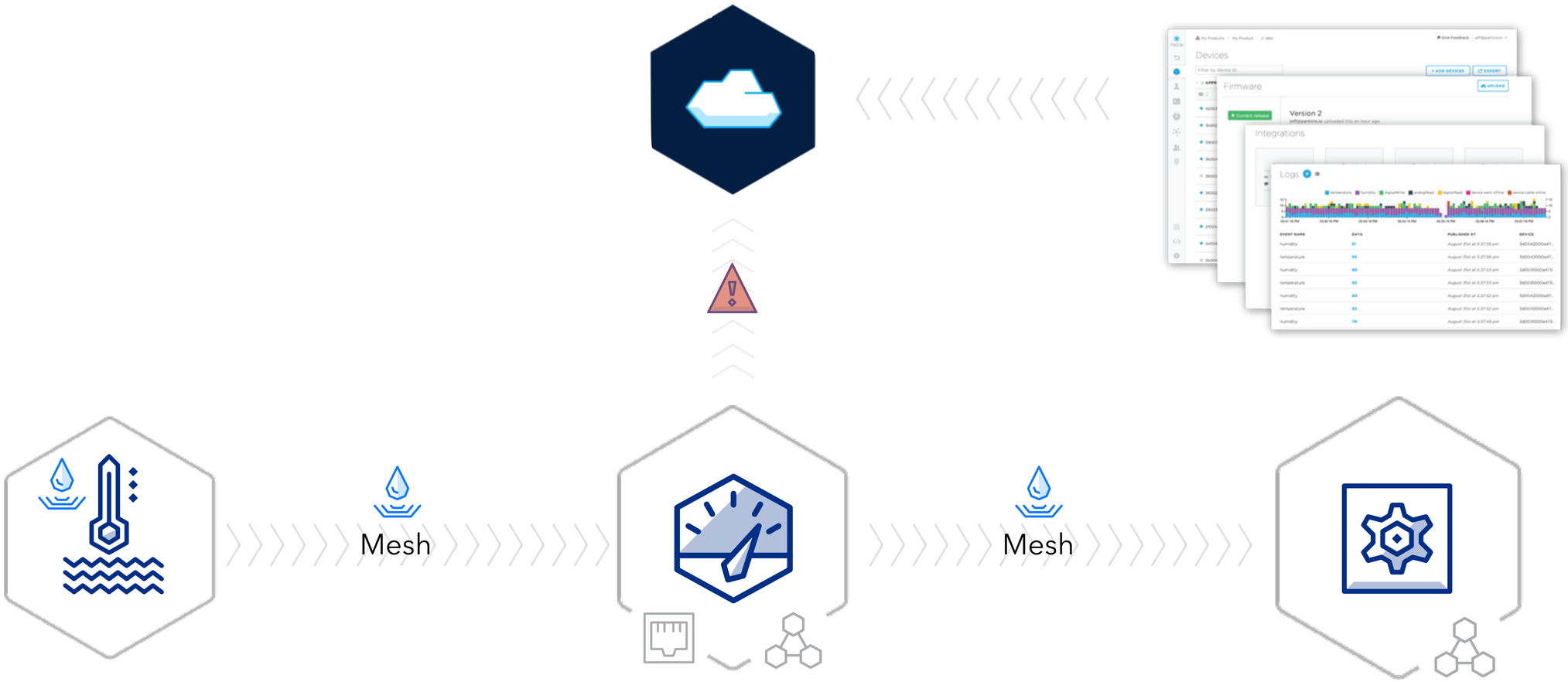
PUB/SUB WITH PARTICLE MESH



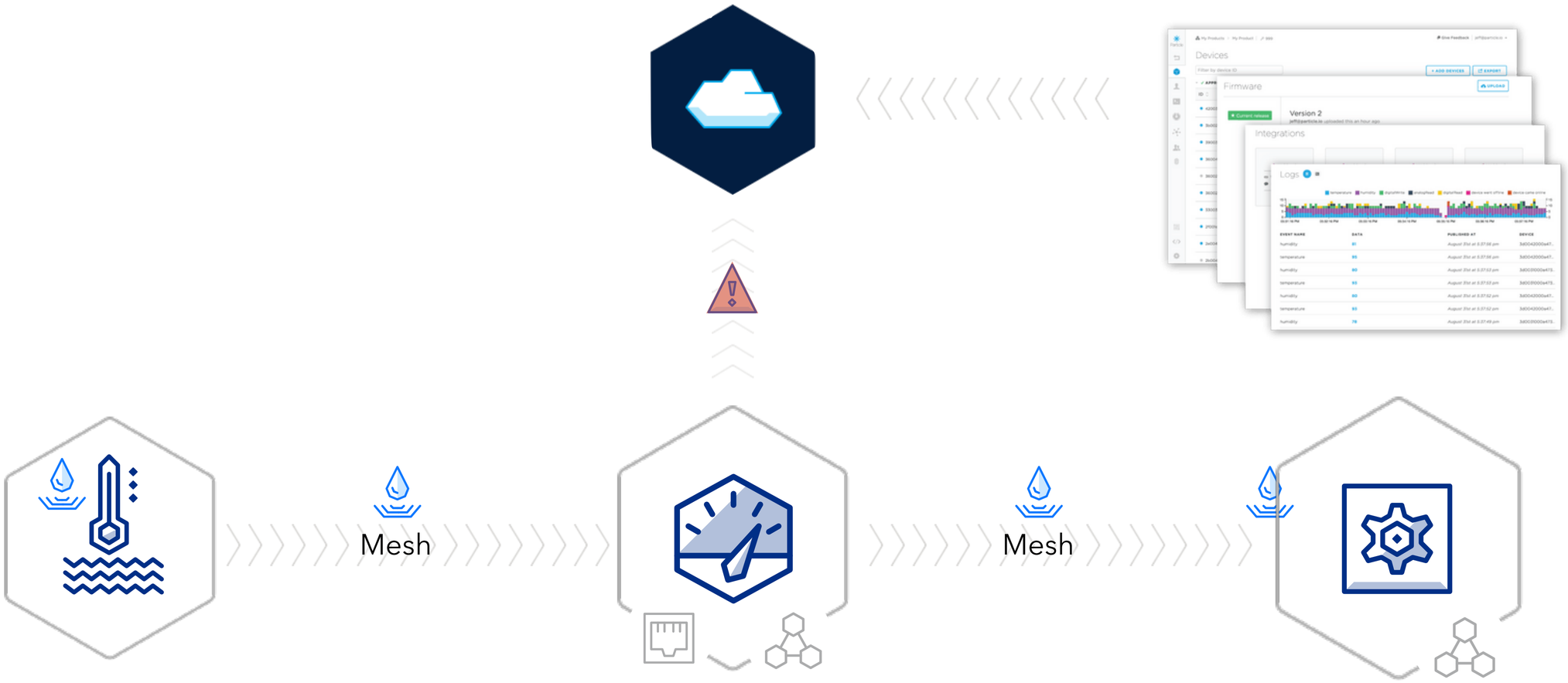
PUB/SUB WITH PARTICLE MESH



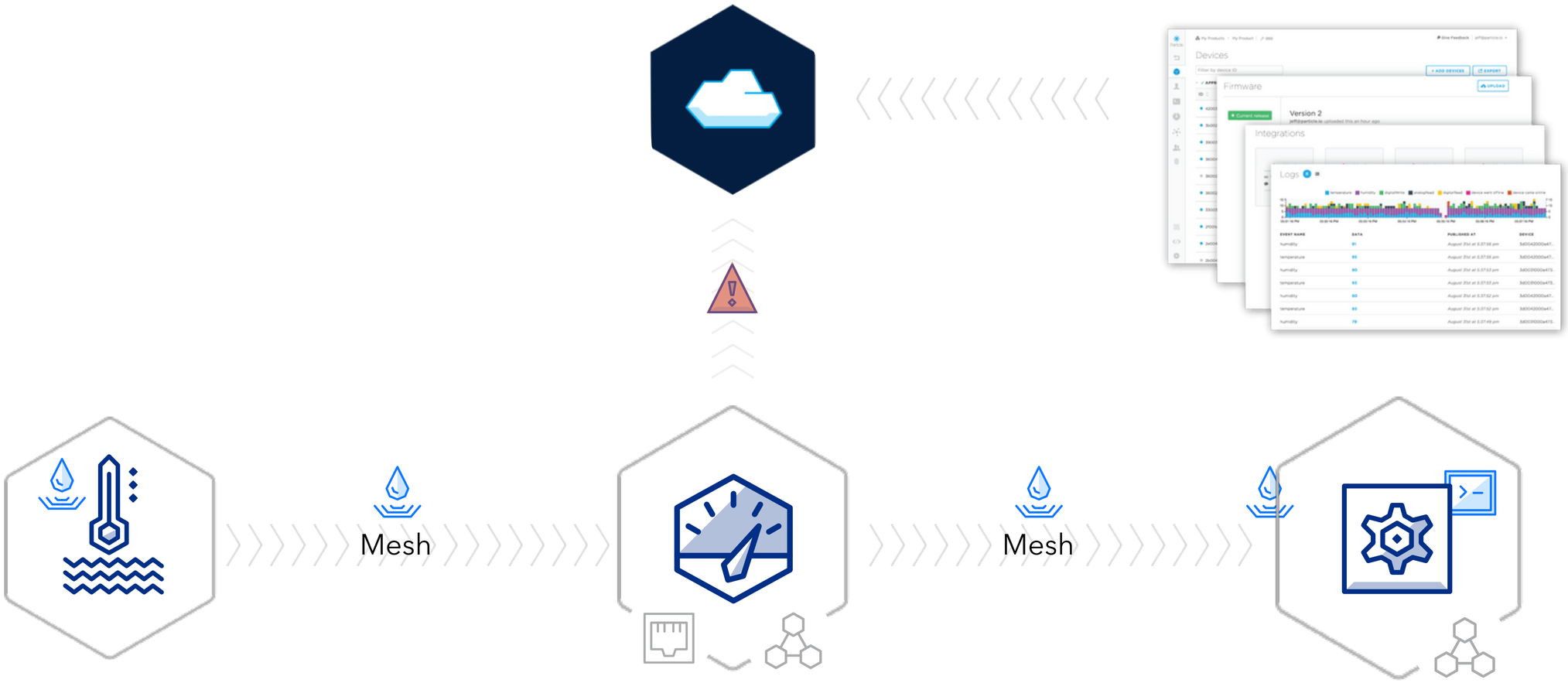
PUB/SUB WITH PARTICLE MESH



PUB/SUB WITH PARTICLE MESH



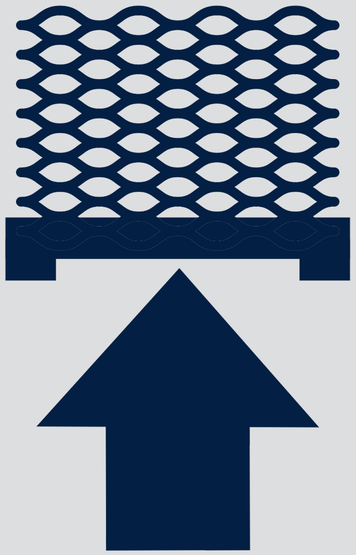
PUB/SUB WITH PARTICLE MESH



PARTICLE MESH FUNCTIONS

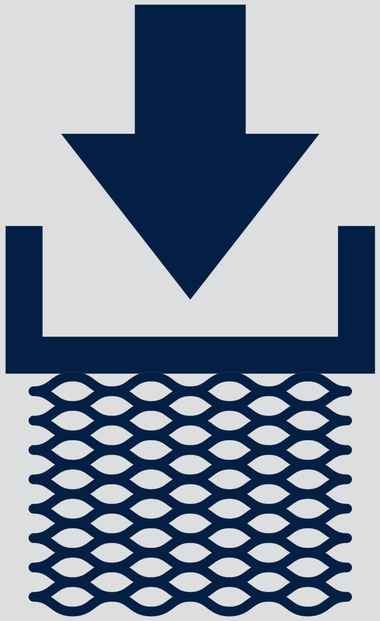
Broadcast an event to all devices in a Mesh network

Mesh.publish()



Listen for events published to the Mesh network

Mesh.subscribe()



MESH.PUBLISH()

What it does:

Publish an event that will be forwarded to all registered listeners on the local Particle mesh network.

Why it's cool:

- Enables mesh network communication
- Works even when the network isn't connected to the cloud

Usage notes:

- 63 characters max for event names

```
double tempC = 0;

void setup()
{
  Particle.variable("temp", tempC);

  pinMode(A0, INPUT);
}

void loop()
{
  analogvalue = analogRead(A0);
  tempC = (((analogvalue * 3.3) / 4095) - 0.5) * 100;

  if (tempC > 120)
  {
    Mesh.publish("temp/critical", tempC);
  }
  else if (tempC > 80)
  {
    Mesh.publish(“temp/warning", tempC);
  }
}
```

MESH.SUBSCRIBE()

What it does:

Subscribe to events published by devices on the local mesh network.

Why it’s cool:

- Enables mesh network communication
- Works even when the network isn’t connected to the cloud

Usage notes:

- Subscriptions work like prefix filters, meaning you can capture multiple publish events via clever naming.

```
void setup()
{
  // Subscribes to temp/warning AND temp/critical
  Mesh.subscribe(“temp", handleTemp);
}

void handleTemp(const char *event, const char *data)
{
  double temp = extractTemp(data);

  if (temp > 120)
  {
    deactivatePump();
  }
  else if (temp > 80)
  {
    reducePumpSpeed();
  }
}
```

GETTING STARTED WITH THE WEB IDE

MESH PUBLISH & SUBSCRIBE

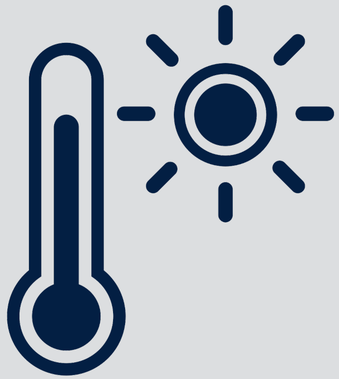
WORKING WITH PARTICLE PRIMITIVES

MANAGING DEVICES FROM THE CONSOLE

PARTICLE CLOUD FUNCTIONS

Call a function, remotely

Particle.function()

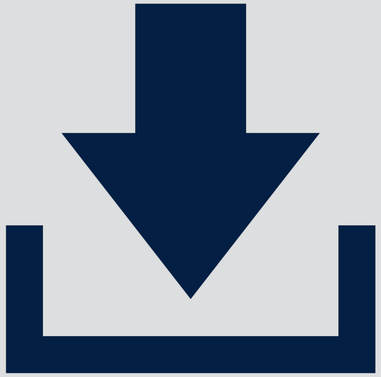


Fetch a variable, remotely

Particle.variable()

Send an event to the cloud

Particle.publish()



Listen for events

Particle.subscribe()

PARTICLE.VARIABLE()

What it does:

Expose a firmware variable to the cloud

Why it's cool:

- Can be fetched via the Device Cloud API
- Viewable from the Device Console

Usage notes:

- 20 variables max.
- 12 character limit per variable name

```
int analogvalue = 0;
double tempC = 0;

void setup()
{
  // variable name max length is 12 characters long
  Particle.variable("analogvalue", analogvalue);
  Particle.variable("temp", tempC);

  // Setup for Sensor on A0
  pinMode(A0, INPUT);
}

void loop()
{
  // Read the analog value of the sensor
  analogvalue = analogRead(A0);

  //Convert the reading into degrees Celsius
  tempC = (((analogvalue * 3.3)/4095) - 0.5) * 100;
  delay(200);
}
```

PARTICLE.VARIABLE()

What it does:

Expose a firmware variable to the cloud

Why it's cool:

- Can be fetched via the Device Cloud API
- Viewable from the Device Console

Usage notes:

- 20 variables max.
- 12 character limit per variable name

```
int analogvalue = 0;
double tempC = 0;

void setup()
{
```

```
# EXAMPLE REQUEST IN TERMINAL
# Device ID is 0123456789abcdef
# Your access token is 123412341234
curl "https://api.particle.io/v1/devices/0123456789abcdef/
analogvalue?access_token=123412341234"
curl "https://api.particle.io/v1/devices/0123456789abcdef/
temp?access_token=123412341234"

# In return you'll get something like this:
960
27.44322344322344
```

```
  //Convert the reading into degrees Celsius
  tempC = (((analogvalue * 3.3)/4095) - 0.5) * 100;
  delay(200);
}
```

PARTICLE.FUNCTION()

What it does:

Expose a firmware function to the cloud

Why it's cool:

- Can be called via the Device Cloud API
- Callable from the Device Console

Usage notes:

- 15 functions max.
- 12 character limit per function name

```
int togglePump(String command);

void setup()
{
  // register the cloud function
  Particle.function("togglePump", togglePump);
}

// this function automagically gets called upon a matching
POST request
int togglePump(String command)
{
  if (command == "on")
  {
    activateWaterPump();
  }
  else
  {
    deactivatePump();
  }

  return 1;
}
```

PARTICLE.FUNCTION()

What it does:

Expose a firmware function to the cloud

Why it's cool:

- Can be called via the Device Cloud API
- Callable from the Device Console

Usage notes:

- 15 functions max.
- 12 character limit per function name

```
int togglePump(String command);

void setup()
{
```

```
# API Call
# GET /v1/devices/{DEVICE_ID}/{VARIABLE}

# EXAMPLE REQUEST IN TERMINAL
# Device ID is 0123456789abcdef
# Your access token is 123412341234
curl "https://api.particle.io/v1/devices/0123456789abcdef/
analogvalue?access_token=123412341234"
curl "https://api.particle.io/v1/devices/0123456789abcdef/
temp?access_token=123412341234"

# In return you'll get something like this:
960
27.44322344322344
```

```
  }

  return 1;
}
```

PARTICLE.PUBLISH()

What it does:

Publish an event that will be forwarded to all registered listeners.

Why it's cool:

- Enables device-to-device communication
- Viewable from the Device Console

Usage notes:

- 63 characters max for event names
- Events are public by default, but can be marked as private.

```
double tempC = 0;

void setup()
{
  Particle.variable("temp", tempC);

  pinMode(A0, INPUT);
}

void loop()
{
  analogvalue = analogRead(A0);
  tempC = (((analogvalue * 3.3) / 4095) - 0.5) * 100;

  if (tempC > 120)
  {
    Particle.publish("temp/critical", tempC);
  }
  else if (tempC > 80)
  {
    Particle.publish("temp/warning", tempC);
  }
}
```

PARTICLE.PUBLISH()

What it does:

Publish an event that will be forwarded to all registered listeners.

Why it's cool:

- Enables device-to-device communication
- Viewable from the Device Console

Usage notes:

- 63 characters max for event names
- Events are public by default, but can be marked as private.

```
double tempC = 0;

void setup()
{
```

```
# API Call
# GET /v1/events/{EVENT_NAME}

# EXAMPLE REQUEST
curl -H "Authorization: Bearer {ACCESS_TOKEN_GOES_HERE}" \
https://api.particle.io/v1/events/temp/critical

# Will return a stream that echoes text when your event is
published
event: temp/critical
data:
{"data":"125","ttl":"60","published_at":"2018-05-28T19:20:34
.638Z",
  "deviceid":"0123456789abcdef"}
```

```
    Particle.publish("temp/warning", tempC);
  }
}
```

PARTICLE.SUBSCRIBE()

What it does:

Subscribe to events published by devices.

Why it's cool:

- Enables device-to-device communication
- Non-IoT devices can also trigger events

Usage notes:

- 4 subscribe handlers per device, max
- Subscriptions work like prefix filters, meaning you can capture multiple publish events via clever naming.

```
void setup()
{
  // Subscribes to temp/warning AND temp/critical
  Particle.subscribe("temp", handleTemp);
}

void handleTemp(const char *event, const char *data)
{
  double temp = extractTemp(data);

  if (temp > 120)
  {
    deactivatePump();
  }
  else if (temp > 80)
  {
    reducePumpSpeed();
  }
}
```

LOCAL MESH PUB/SUB VS. PARTICLE CLOUD PUB/SUB

Mesh Pub/Sub is for local messages

Use Mesh Pub/Sub When:

- You need to communicate between devices *only* on a mesh
- You need messages to be sent as fast as possible
- You need to communicate between devices when a connection to the cloud is unavailable.
- It's ok that not *every* message is delivered.

Particle Pub/Sub is for everything else

Use Particle Pub/Sub When:

- You need to communicate between mesh networks or with devices not on a mesh network
- You're publishing events to webhooks or cloud integrations (Azure, Google Cloud, etc.)
- You need some QOS in message delivery (retry attempts, etc.)

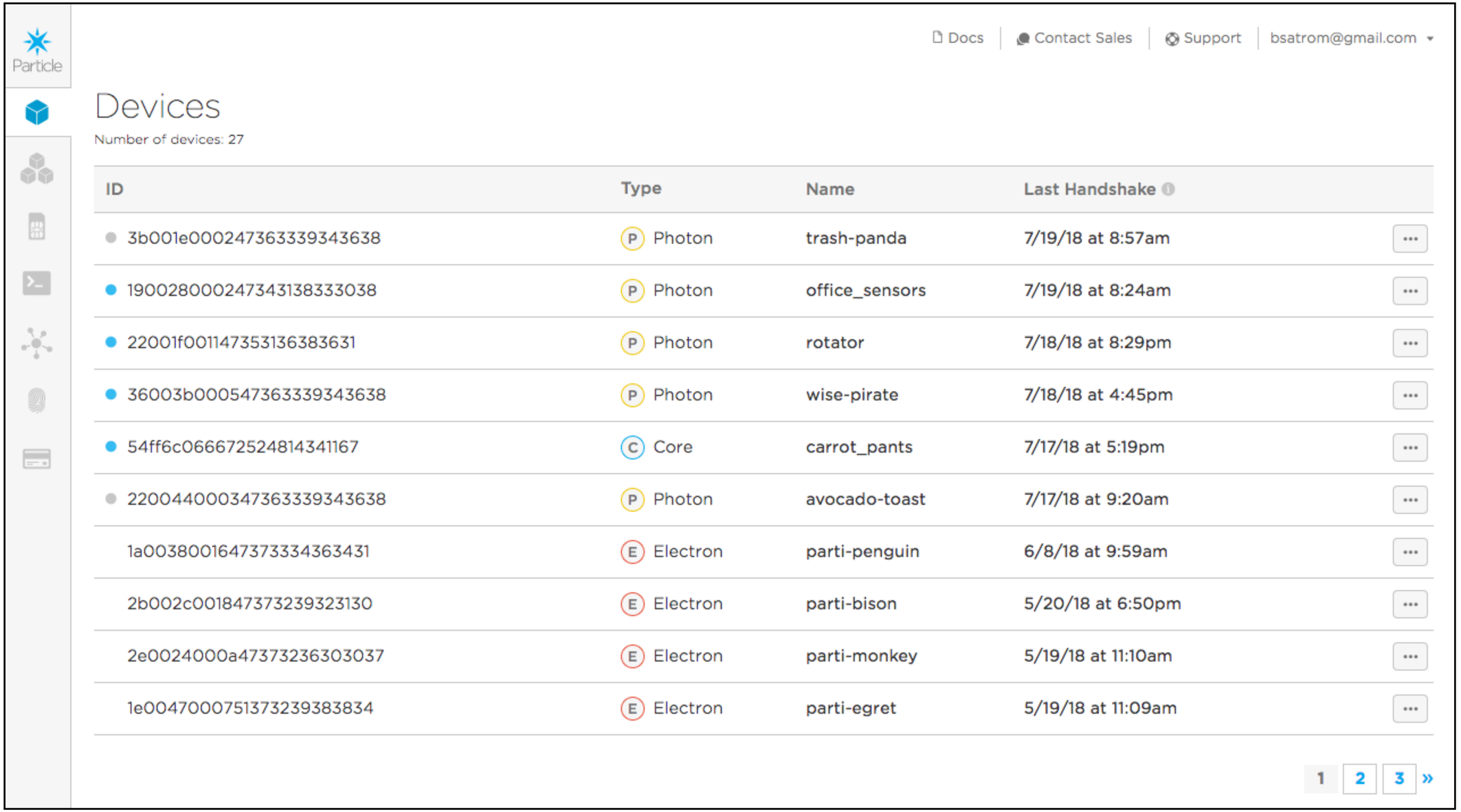
GETTING STARTED WITH THE WEB IDE

MESH PUBLISH & SUBSCRIBE

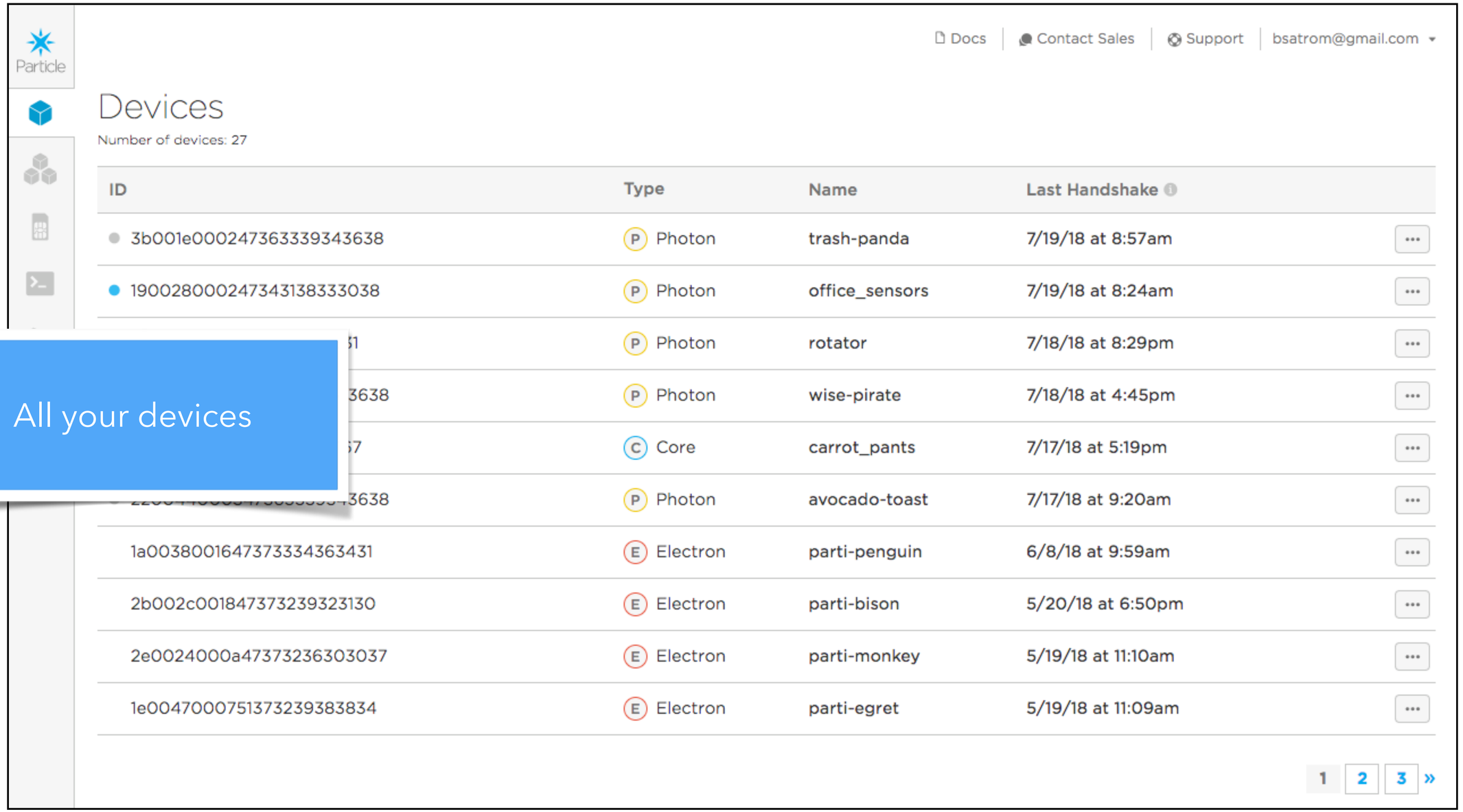
WORKING WITH PARTICLE PRIMITIVES

MANAGING DEVICES FROM THE CONSOLE

CONSOLE.PARTICLE.IO

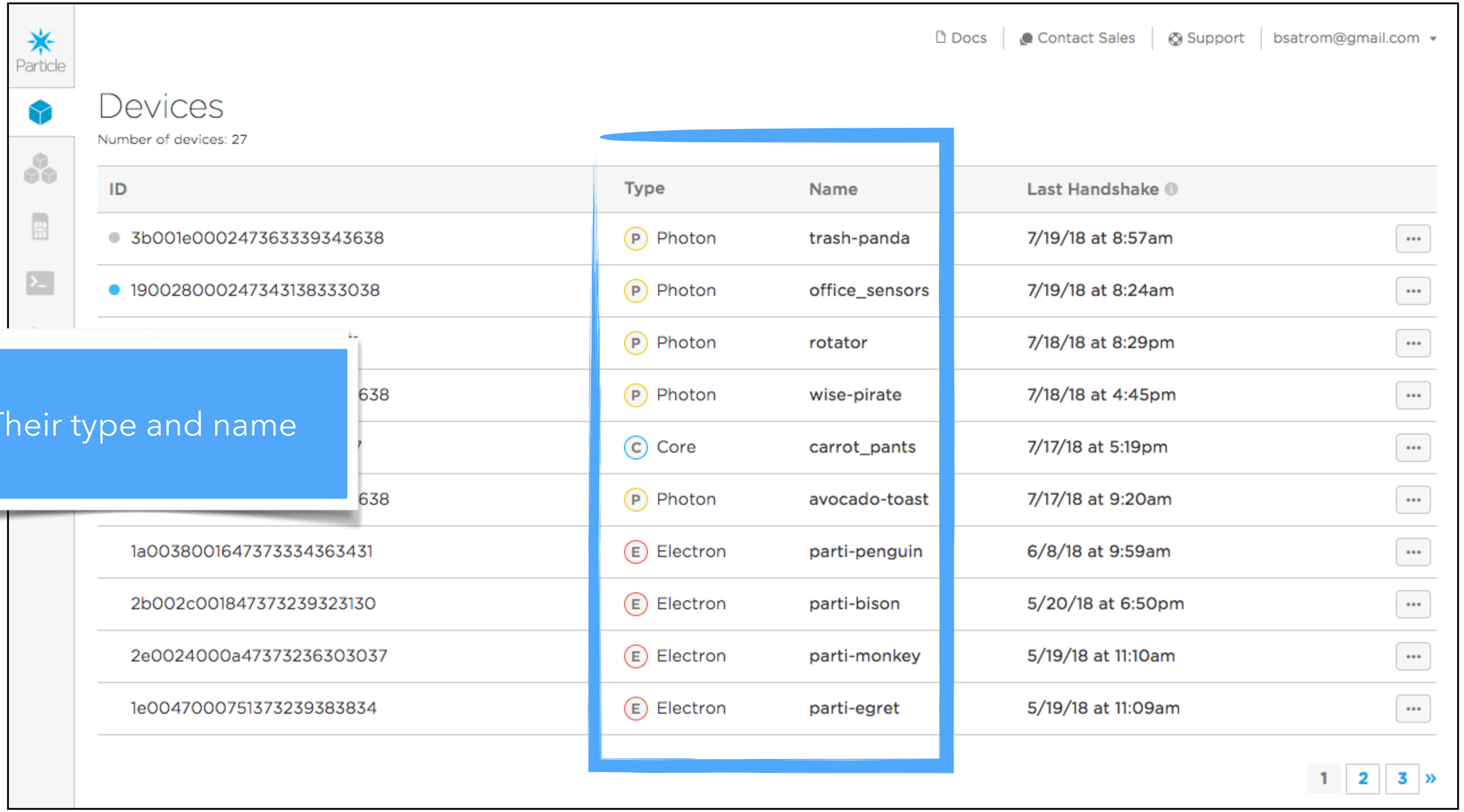


CONSOLE.PARTICLE.IO

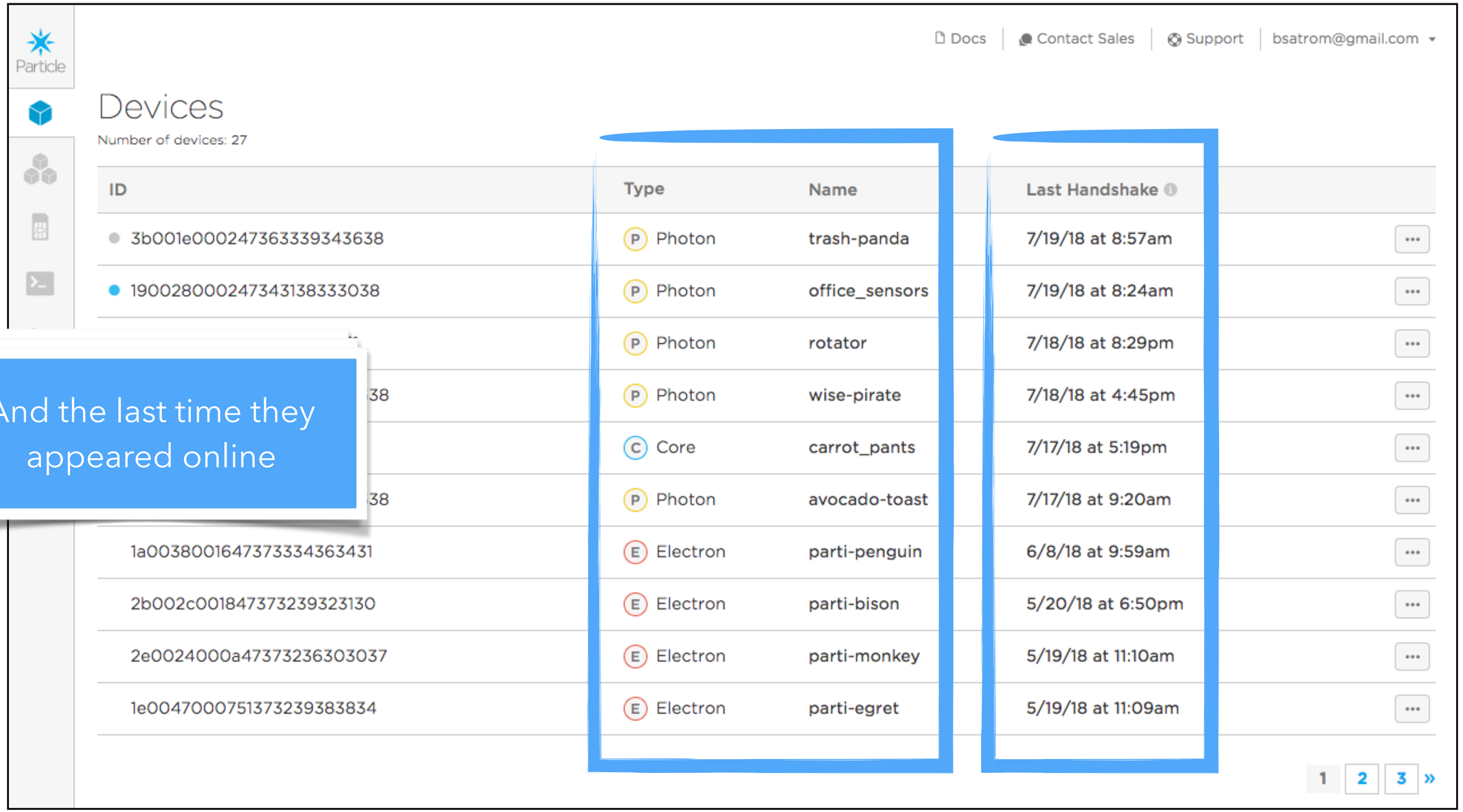


All your devices

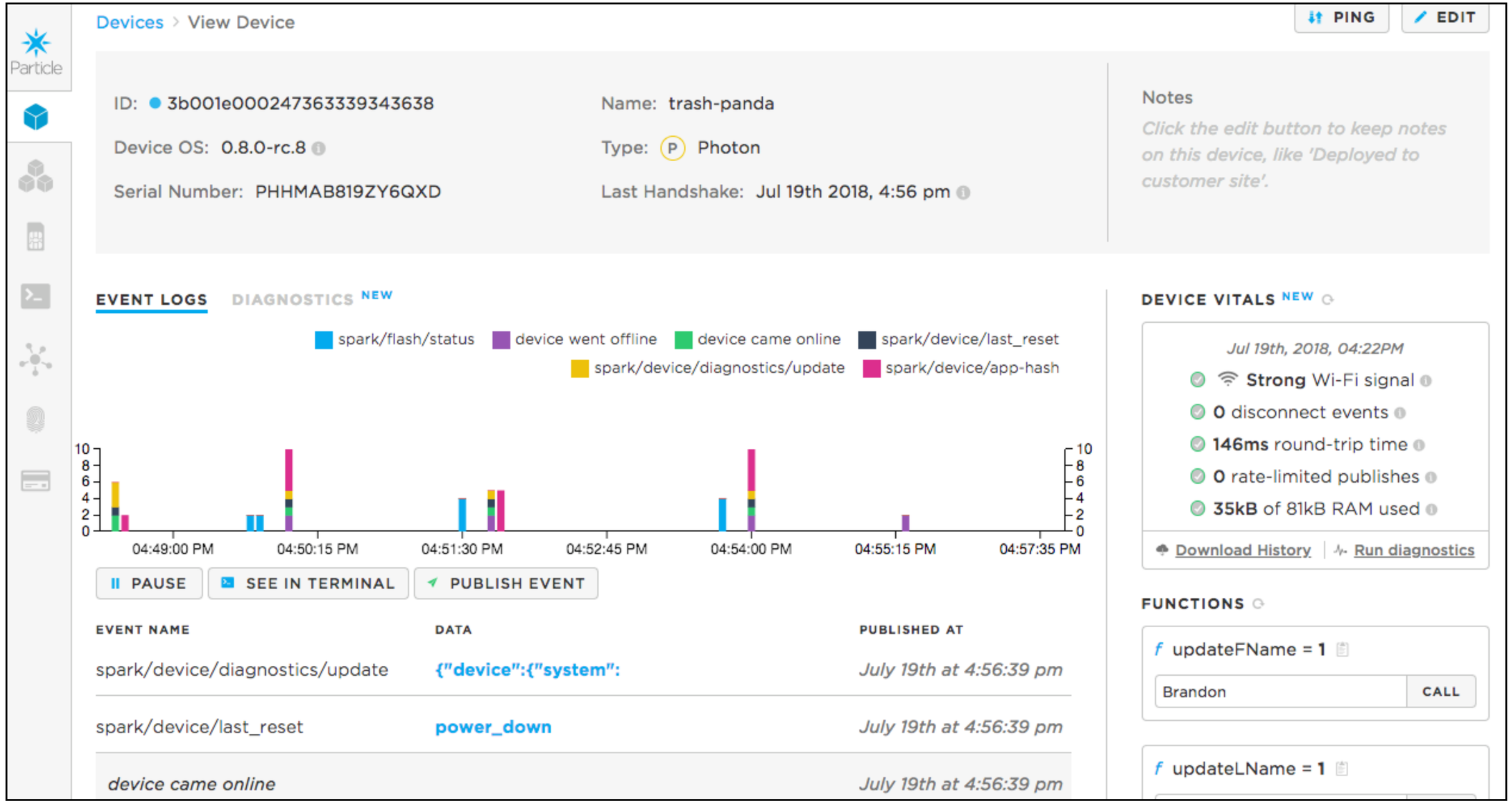
CONSOLE.PARTICLE.IO



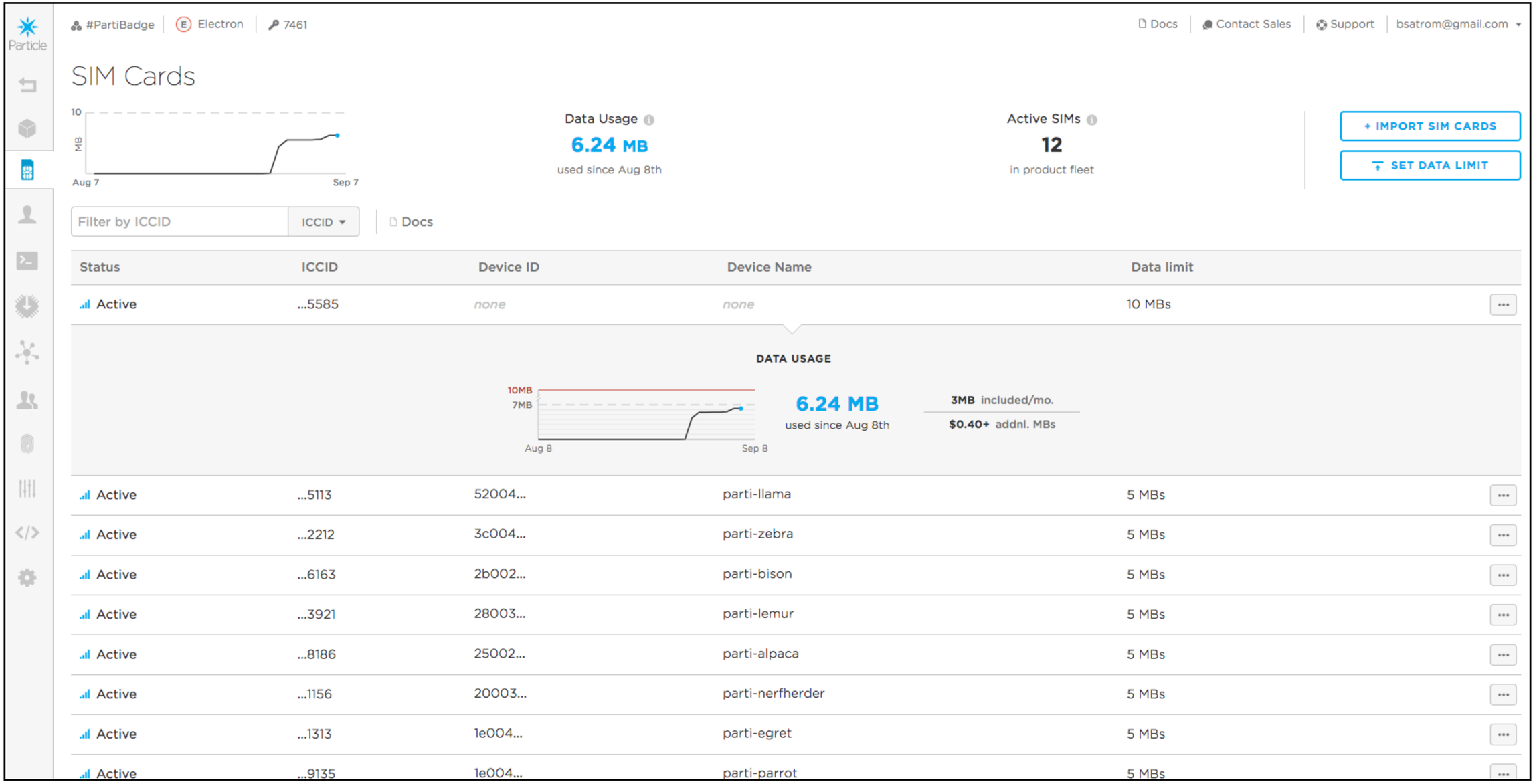
CONSOLE.PARTICLE.IO



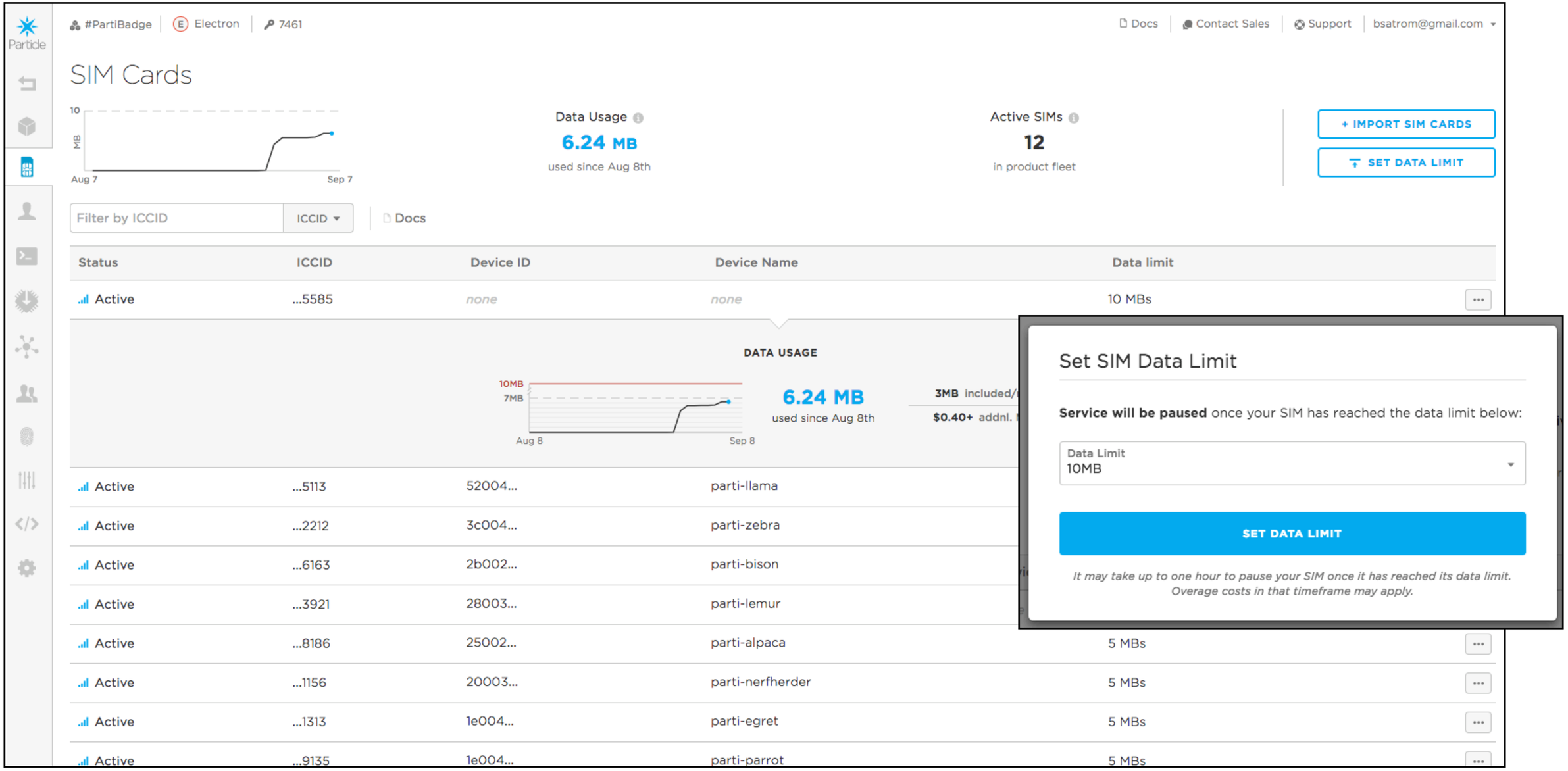
REAL-TIME EVENT LOGS



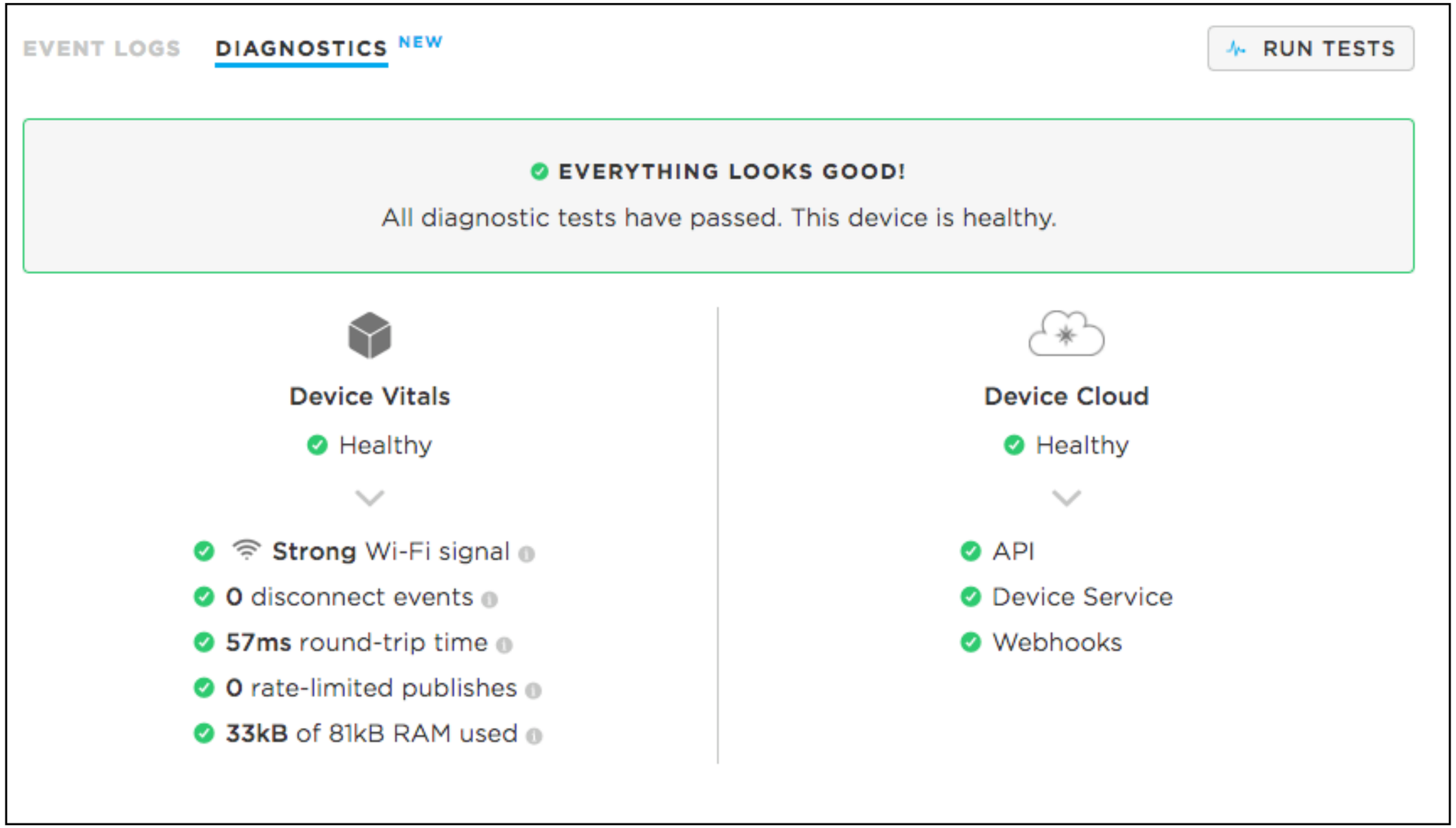
SIM MANAGEMENT



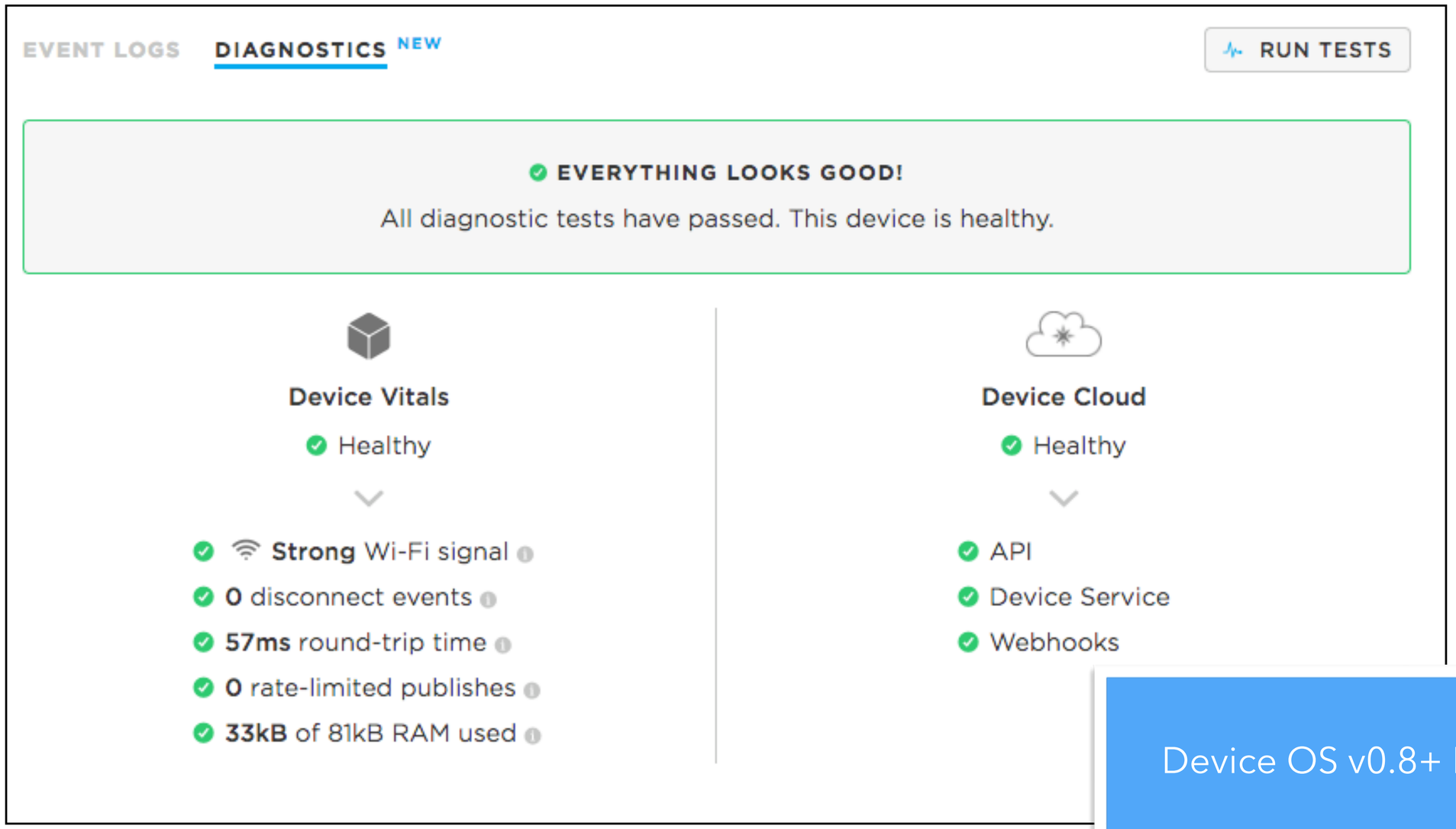
SIM MANAGEMENT



REMOTE DIAGNOSTICS

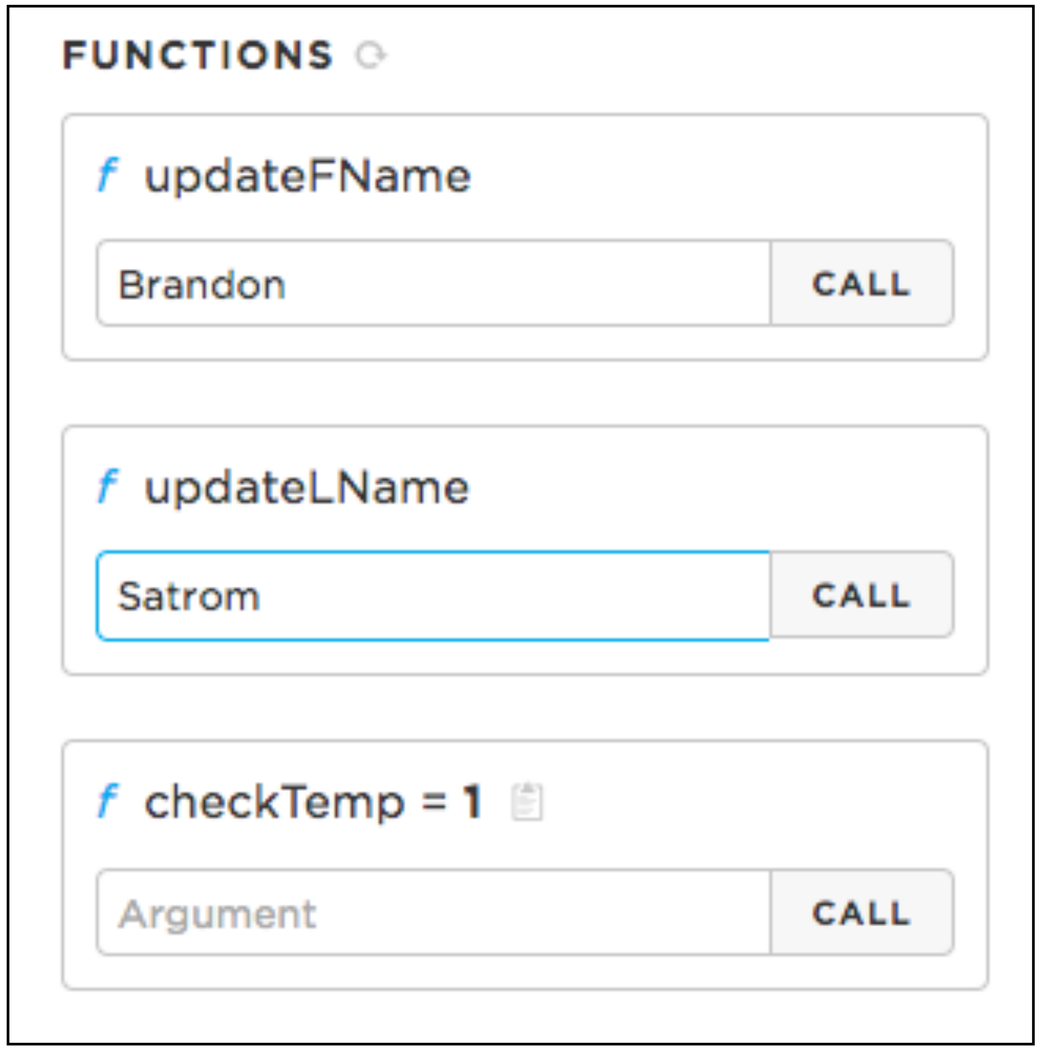


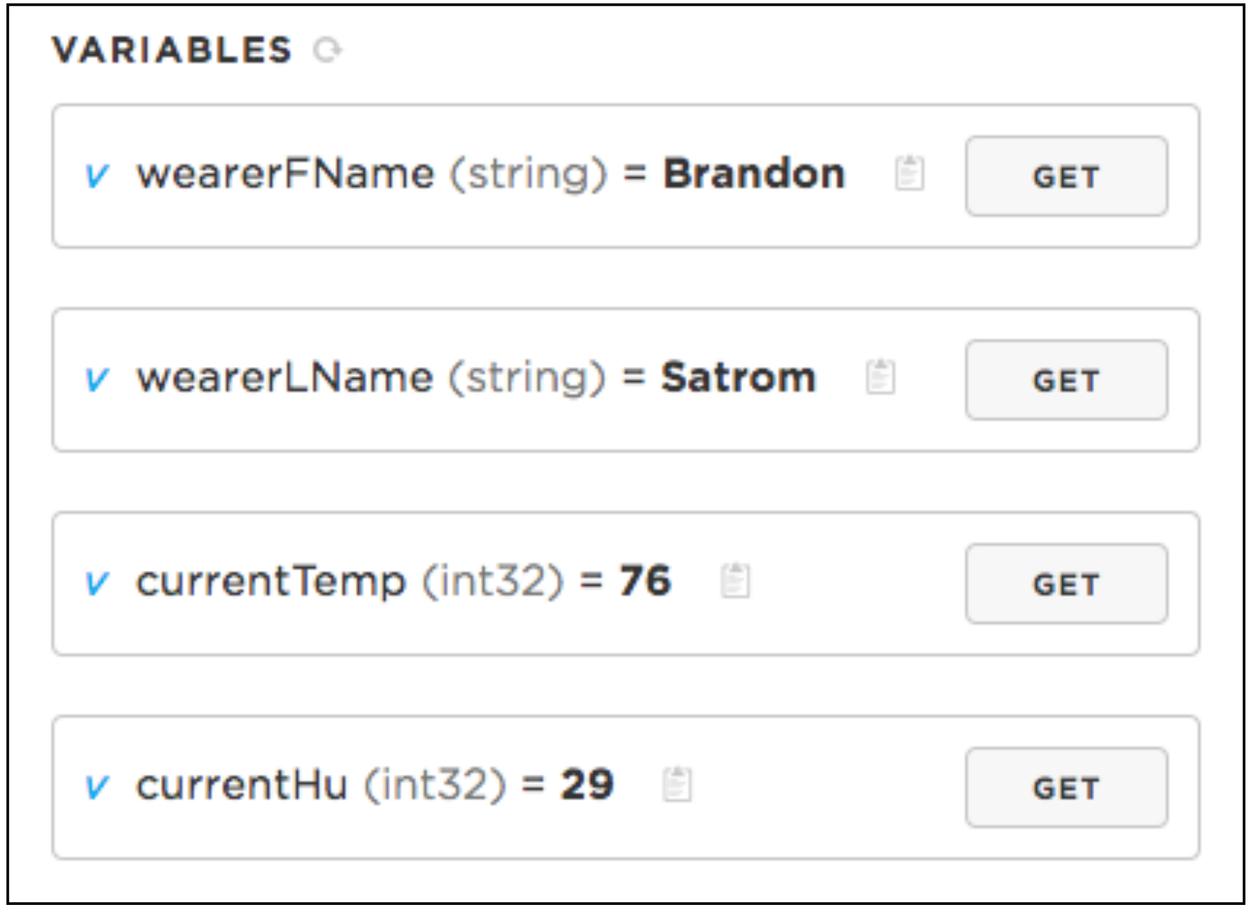
REMOTE DIAGNOSTICS

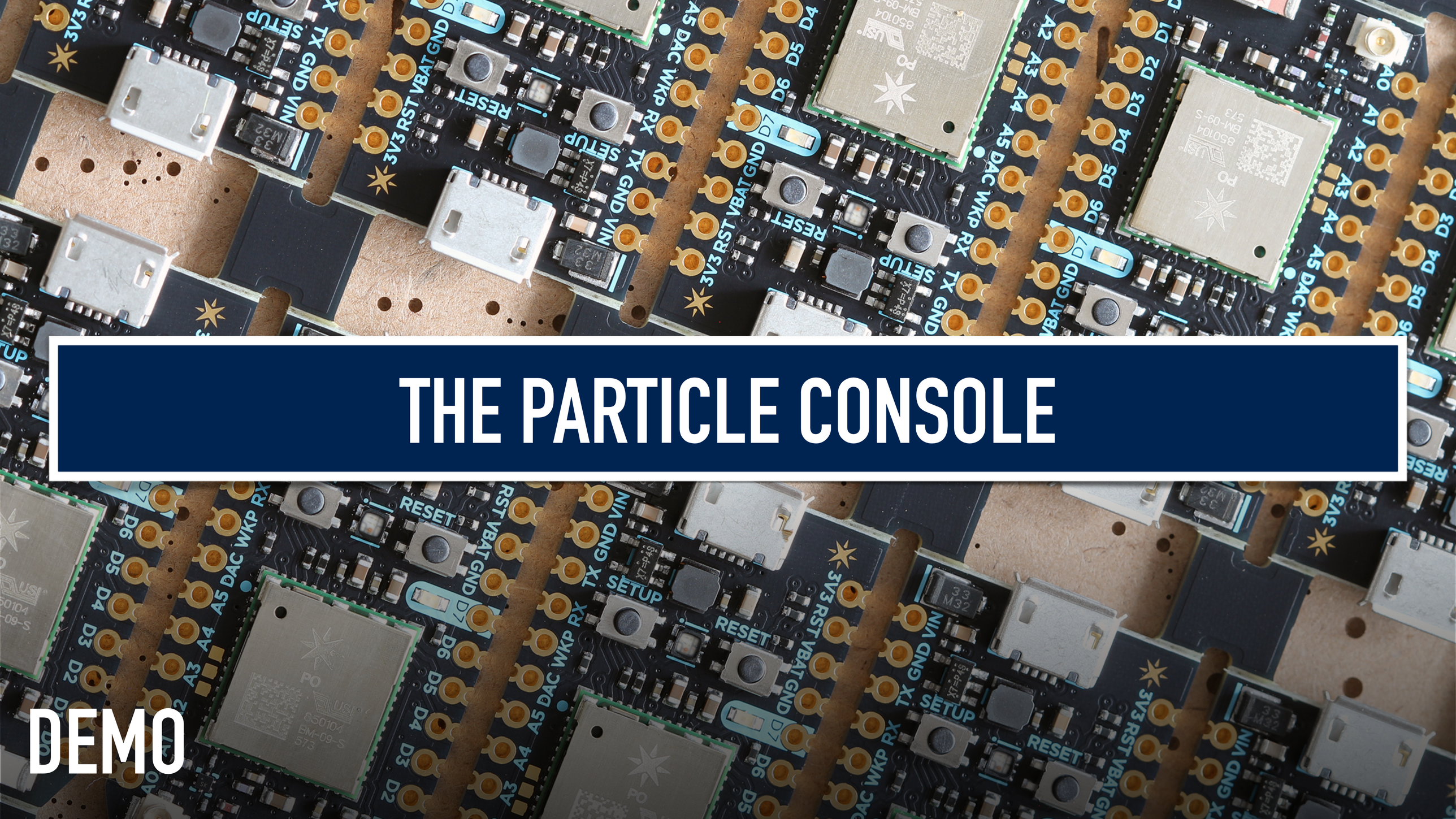


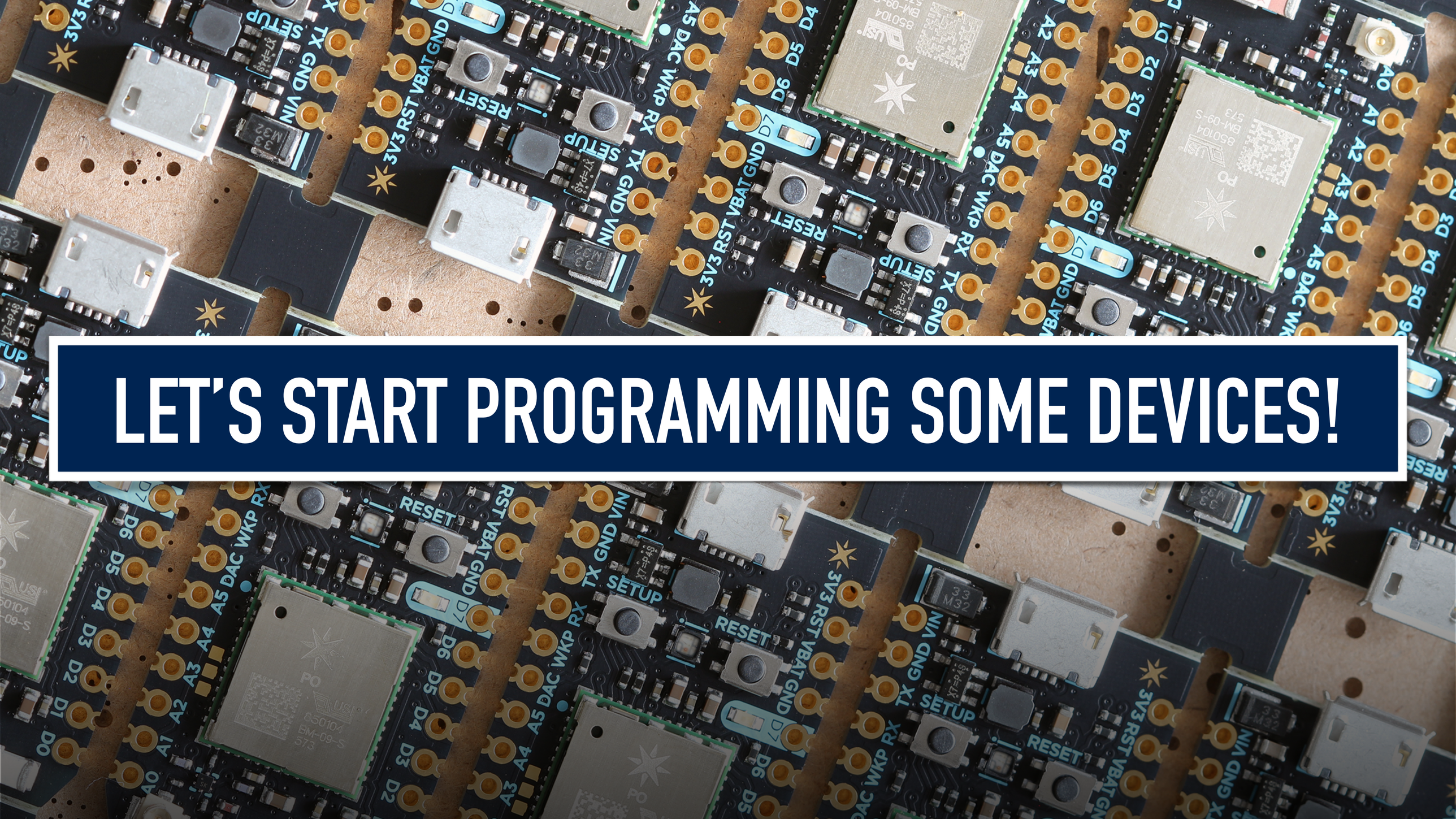
Device OS v0.8+ Required!

VIEWING CLOUD VARIABLES AND CALLING CLOUD FUNCTIONS






THE PARTICLE CONSOLE
DEMO


LET'S START PROGRAMMING SOME DEVICES!